



8048 Family Applications Handbook

Q. **Quantum Electronics**
 Jan 3 + 1962
 Bramley
 2018

©Intel Corporation, January 1980

\$5.00

8048 Family Applications Handbook

The following are trademarks of Intel Corporation and may be used only to describe Intel products: Intel, Insite, Inteltec, Library Manager, Mega-chassis, Micromap, Multibus, PROMPT, RMX/80, UPI, Intelevison, μ Scope, Promware, MCS, ICE, HSE, ISBC, BXP, ICS, and the combination of MCS, ICE, ISBC or ICS with a numerical suffix.

Intel Corporation Assumes No Responsibility for the Use of Any Circuitry Other Than Circuitry Embodied in an Intel Product. No Other Circuit Patent Licenses are Implied.

Table of Contents

Introduction	iii
--------------------	-----

MCS-48™ FAMILY

Single-Chip 8-Bit Microcomputer fills gap between Calculator Types and Powerful Multichip Processors ..	1-1
Application Techniques for the MCS-48 Family	2-1
Keyboard/Display Scanning with Intel's MCS-48 Microcomputers	3-1
Serial I/O and Math Utilities for the 8049 Microcomputer	4-1
A High-Speed Emulator for Intel MCS-48 Microcomputers	5-1
Microcontroller Includes A-D Converter for Lowest-Cost Analog Interfacing	6-1
Microcomputer's On-Chip Functions Ease Users' Programming Chores	7-1
Designing with Intel's 8022 Microcomputer	8-1

UPI-41

Introduction to the UPI-41A	9-1
Printer Control with the UPI-41	10-1
Using the 8295 Dot Matrix Printer Controller	11-1

GENERAL

Application of Intel's 5V EPROM and ROM Family for Microprocessor Systems	12-1
CRYSTALS: Specifications for Intel Components	13-1

INTRODUCTION

The MCS-48™ family of single-chip-microcomputers has become an industry standard since the introduction of its original member (the 8748) in 1979. The family is now comprised of seven members (see table). All of these components share a common architecture; each of them has unique features which may prove beneficial in a given application.

This manual is a collection of the application information available for the MCS-48 family. Several items concerning Intel's UPI-41™ family are also included. The UPI-41 family is a series of universal peripheral interface devices which have an architecture which almost duplicates that of the MCS-48. The only significant difference is that UPI-41

devices reside on a system bus as a slave device whereas MCS-48 components are typically bus masters. Because of the similarity between these two series of parts, application techniques can usually be applied equally well to members of both families. It is hoped that the inclusion of the application notes concerning the UPI-41 family will be useful to designers working with MCS-48 family components.

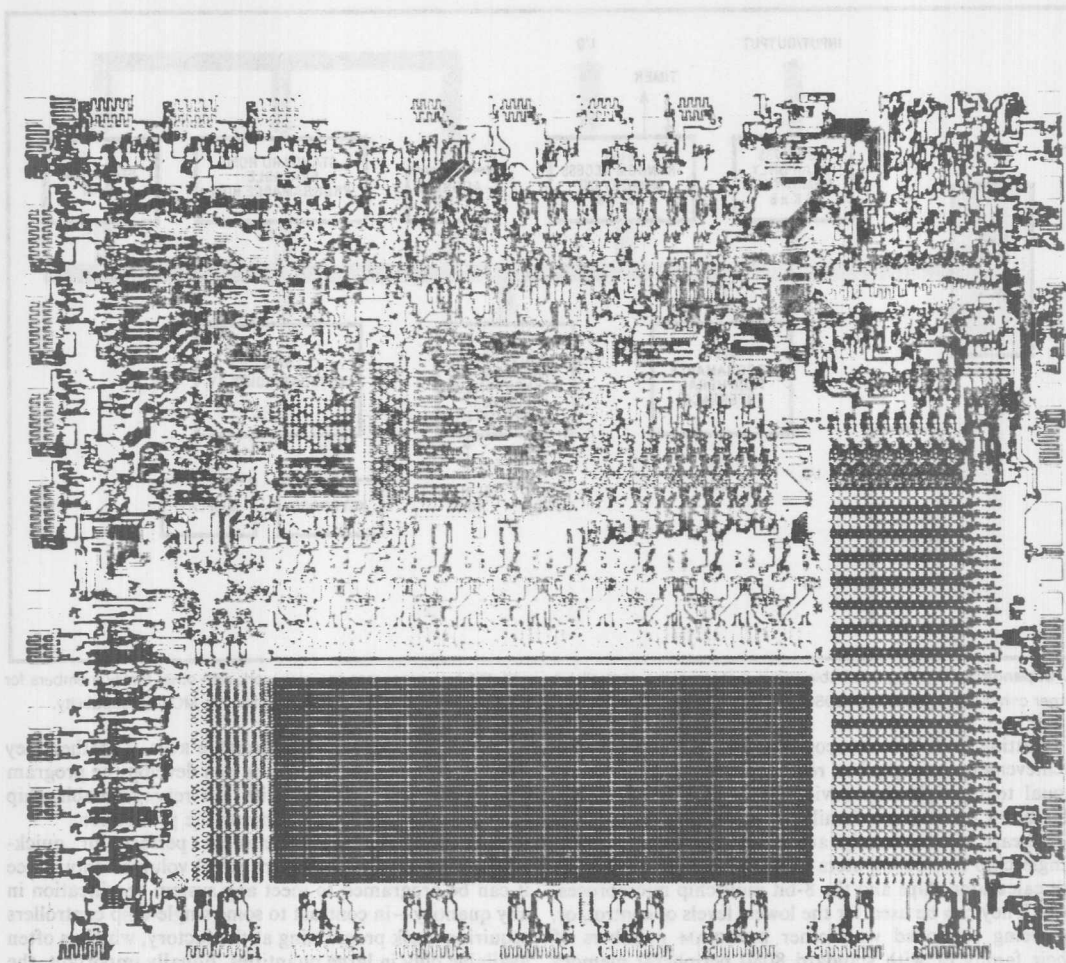
The material included in this manual is believed to be accurate; if you find any errors, or if you have any suggestions for future application notes for the MCS-48 family, we would appreciate hearing from you.

CHARACTERISTICS	COMPONENT						
	8021	8022	8035	8039	8048	8049	8748
ROM SIZE (KILOBYTES)	1	2	—	—	1	2	1*
RAM SIZE (BYTES)	64	64	64	128	64	128	64
I/O PINS	21	28	15	15	27	27	27
CYCLE TIME (MICRO SEC)	8.5	8.5	2.5	1.4	2.5	1.4	2.5
A/D CHANNELS	—	2	—	—	—	—	—
*Erasable EPROM							

December 1976

**Single-chip 8-bit microcomputer fills
gap between calculator types and
powerful multichip processors**

H. Blume, D. Budde, B. Morgan, H. Raphael, P. Salisbury, D. Stamm
Electronics November 25, 1976



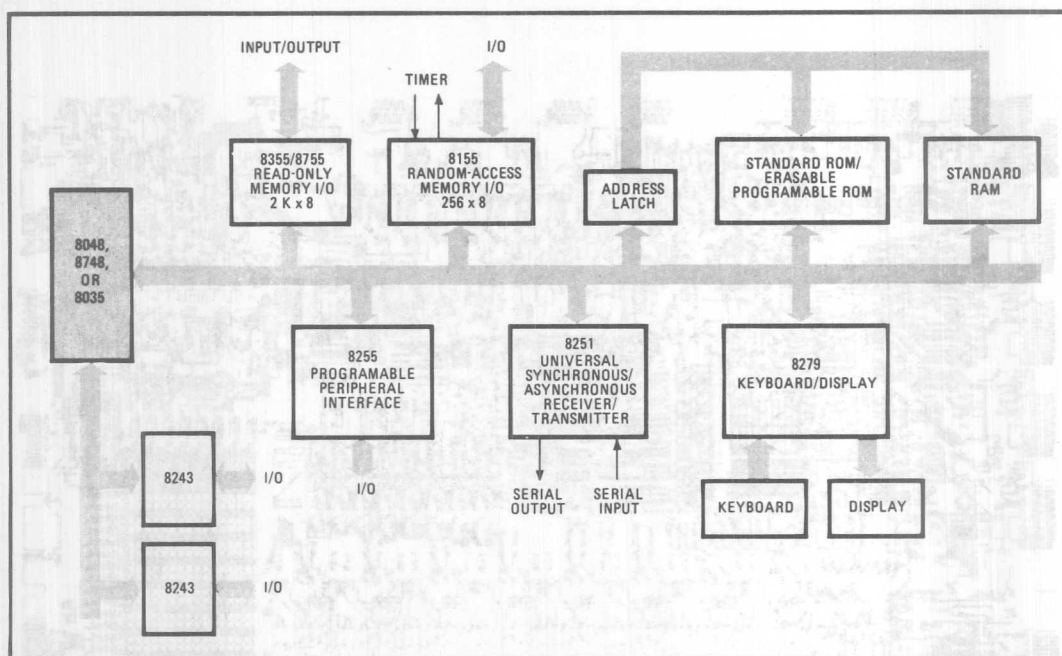
© Intel 1976

Single-chip 8-bit microcomputer fills gap between calculator types and powerful multichip processors

Capabilities range from stand-alone computing to high-power data processing; ultraviolet light erases programmable ROM of one version

by Henry Blume, David Budde, Bill Morgan, Howard Raphael, Phil Salsbury, David Stamm,

Intel Corp., Santa Clara, Calif.



1. Expandable. Although well able to run a stand-alone controller by itself, the new processor can also work with other family members for larger control systems or with 8080 peripherals to handle complex data processing. This configuration typifies the MCS-48 capability.

□ Putting an 8-bit microcomputer onto a single chip is achievement enough, but realizing performance nearly equal to multiple-chip devices gives a bonus of added flexibility for the new family. The two devices that are the heart of the family are really high-performance, single-chip microcomputers that fill the gap between 4-bit calculator chips and the 8-bit multichip microprocessors. They can be used for the lowest levels of control, or, by being expanded with other ROM/RAM members of their family or with standard 8080 peripheral memory chips, they can be used in a wide range of high-powered data-processing systems.

The two versions of the microcomputer, the 8748 and the 8048, are like 4-bit calculator devices in that they each contain all the elements needed for stand-alone computing—central processing unit, program read-only memory, data random-access memory, input/output interface, plus clocks and timers. Yet they contain these elements in 8-bit configurations that vastly exceed the power of the calculator types and approach 8080 power.

Two ROM versions

The MCS-48 family is the first to offer a microprocessor with an erasable programable ROM, which will prove handy for low-volume applications and those in which periodic update of the program memory is required. The family also has a CPU-only chip, the 8035, which can be used with external memories.

The 8748 has a 2708-type, 8,192-bit EPROM with a program that can be changed by clearing with ultraviolet light and reprogramming electrically in the usual way. The

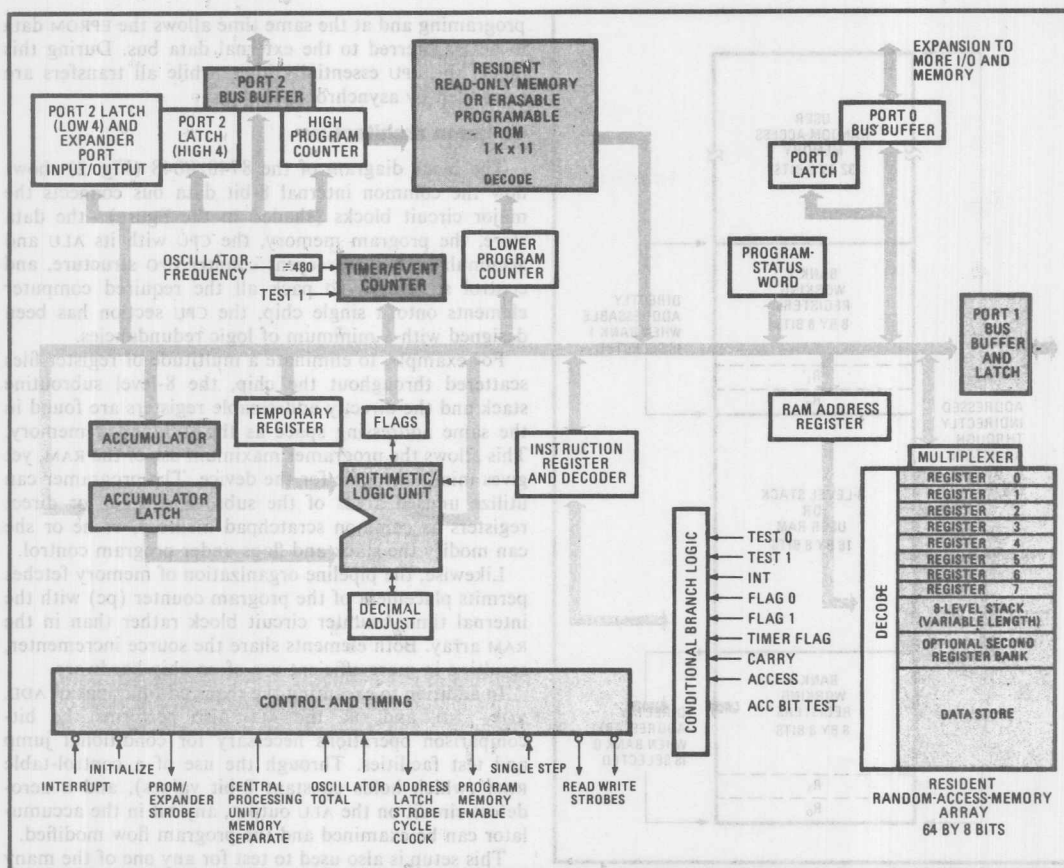
8048 has an 8-k mask-programable ROM. Together they give the user new flexibility: he can develop the program and build the prototypes with the reprogrammable chip and switch to mask ROMs for volume production.

The off-the-shelf 8748 also is perfect for quick-turnaround users who require small volumes only, since it can be programmed to meet any system specification in any quantity—in contrast to some single-chip controllers requiring mask programming at the factory, which is often available only in large quantities. Equally important, the 8748 can be used in control systems requiring periodic updating in the field, such as point-of-sale price-and-inventory controls. New program data can be fed into the system without a new ROM.

The free-standing operations of the 8048 and 8748 are made possible by the 1,024-by-8-bit ROM or EPROM for program memory, a 64-by-9-bit RAM for scratchpad functions, an 8-bit CPU consisting of an arithmetic/logic unit and accumulator for all the binary and decimal arithmetic functions, and an input/output facility that includes three 8-bit I/O ports plus three test/interrupt ports directly controlled by program instructions.

Memory and input/output of the processors can be expanded to handle large control applications (Fig. 1). There's an inexpensive expander chip, 8243, which allows the processor chips to handle an additional 16 I/O lines. Also included in the family are combination memory and I/O expanders, such as a 2,048-by-8-bit ROM with 16 I/O lines (8355), a 2-k-by-8-bit EPROM with 16 I/O lines (8755), and a 256-by-8-bit RAM with 22 I/O lines (8155).

The MCS-48 components also work directly with all



2. Stacked. The 8748 or 8048 processor chip supplies all the functions needed for a stand-alone microcomputer. It has a CPU complete with arithmetic/logic unit and accumulator, a 256-bit RAM, an 8,192-bit program ROM, a timer/event counter, and plenty of I/O capability.

the 8080 family of standard memory and peripheral parts, soon to number about 30 large-scale-integrated circuits. They include timers, programmable I/O controllers, universal synchronous/asynchronous receiver/transmitters, decoders, and keyboard/display controllers.

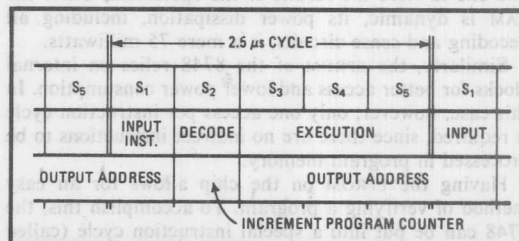
One-chip advantages

The integration of all the basic blocks of a microcomputer system into one circuit brings about some architectural advantages. When the device is used as a stand-alone controller, it need interface only with its I/O peripherals. This means that the execution speed of the processing is limited only by the speed of the chip, because there is no slowdown from transferring data between memory and CPU, as in multiple-chip designs.

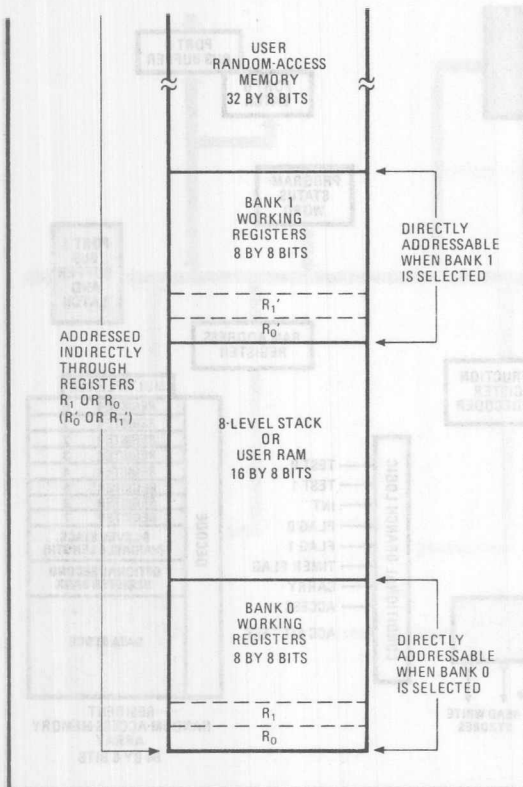
Moreover, technological upgrades can give enhanced performance without waiting for similar upgrades of external components, as is usually the case with multi-chip families. More immediately, the inclusion of data and program memories, which otherwise would have to be added separately to the system, simplifies the user's interface problems.

Having an active data store on the chip—the quasi-static 64-by-8-bit RAM—also simplifies system implementation, since all scratchpad operations simply became part of the CPU function. There is no need for refresh circuits operate the RAM; yet the device is dynamic in the sense that internal clocks are used for very fast, low-power access to the array.

The major objective was access to a RAM within a



3. Simple. Operating the 8748/8048 is extremely straightforward, with each 2.5-μs cycle consisting of five states. Instruction inputs are made in state 1, decoding and program incrementing in state 2. Program executions begin in state 3 and run through 4 and 5.



4. Powerful. The on-chip RAM, part of which is reserved for one or two banks of 8-bit working registers, also accommodates the stack of subroutine addresses, which can be eight levels deep. Each stack location can handle the program counter and status data.

fraction of an instruction cycle, so that those indirect internal instructions that require multiple addresses could still be executed in one instruction cycle. (Indirect RAM instructions require three separate accesses: one to fetch the address of the memory location to be operated on, one to fetch the contents of the addressed location, and one to store the results of the operation.) Since the RAM is dynamic, its power dissipation, including all decoding and sense circuits, is a mere 75 milliwatts.

Similarly, the EPROM of the 8748 relies on internal clocks for better access and lower power consumption. In this case, however, only one access per instruction cycle is required, since there are no indirect instructions to be processed in program memory.

Having the EPROM on the chip allows for an easy method of verifying a program. To accomplish this, the 8748 can be put into a special instruction cycle (called the third-state mode) for programming and verification of the EPROM. The CPU executes a special double-cycle instruction that allows the address and data information to be transferred to their respective registers during

mode, the CPU essentially idles, while all transfers are controlled by asynchronous inputs.

Common architecture

The block diagram of the 8748/8048 (Fig. 2) shows how the common internal 8-bit data bus connects the major circuit blocks (shaded in the figure)—the data store, the program memory, the CPU with its ALU and accumulator, a timer/event counter, I/O structure, and control structure. To pack all the required computer elements onto a single chip, the CPU section has been designed with a minimum of logic redundancies.

For example, to eliminate a multitude of register files scattered throughout the chip, the 8-level subroutine stack and the directly addressable registers are found in the same addressing space as the scratchpad memory. This allows the programmer maximum use of the RAM, yet gives minimum logic for the device. The programmer can utilize unused areas of the subroutine stack or direct registers as common scratchpad memory, or he or she can modify the stack and flags under program control.

Likewise, the pipeline organization of memory fetches permits placement of the program counter (pc) with the internal timer/counter circuit block rather than in the RAM array. Both elements share the source incrementer, resulting in more efficient use of on-chip hardware.

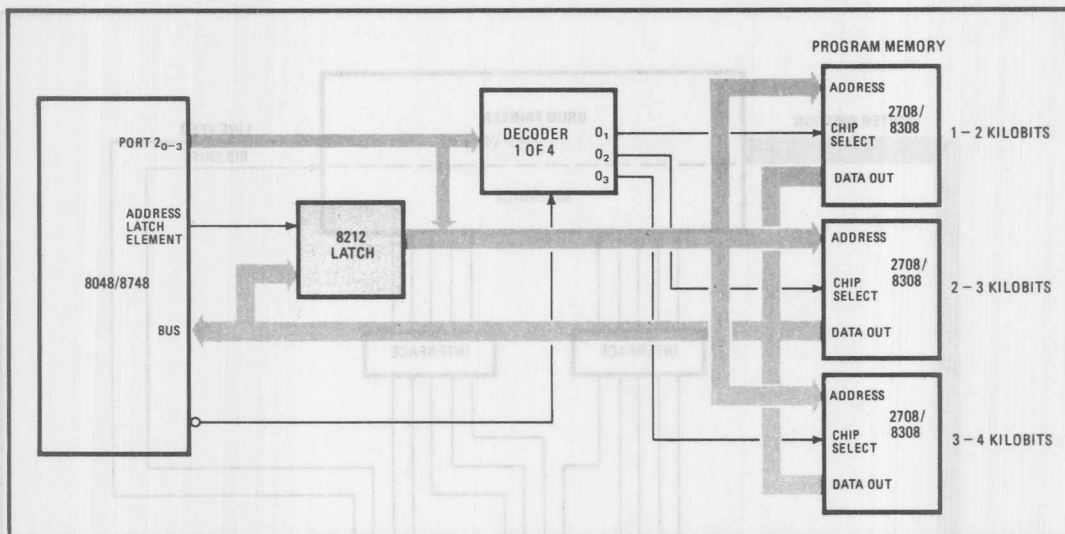
In addition to executing the required functions of ADD, XOR, AND, and OR, the ALU also performs the bit-comparison operations necessary for conditional jump and test facilities. Through the use of a control-table ROM (which holds constant 8-bit values), and a zero-detect circuit on the ALU output, any bit in the accumulator can be examined and the program flow modified.

This setup is also used to test for any one of the many conditional jumps. Each of the conditional-jump flags and inputs is sent to the ALU as an 8-bit conditional word and tested with the same circuitry used to examine individual accumulator bits.

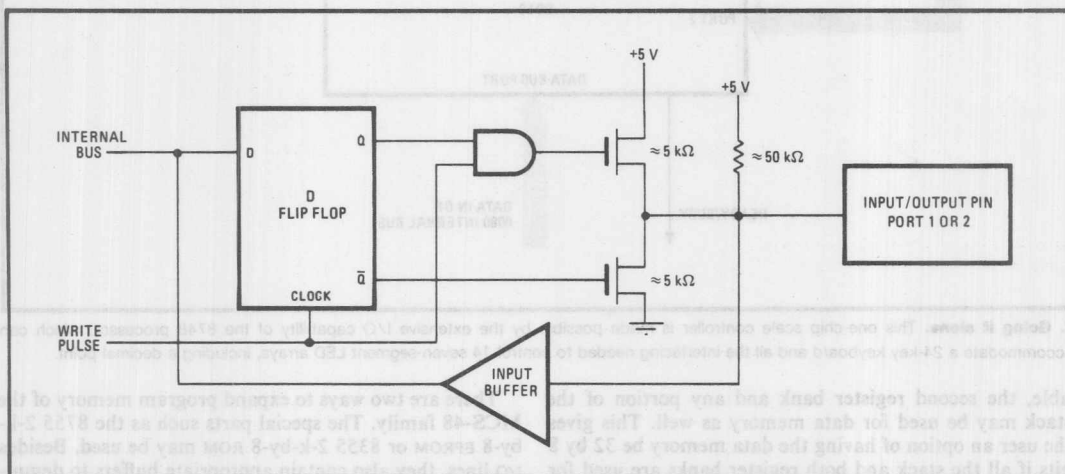
An internal oscillator also gives many system and device savings, such as the elimination of external components (except for a crystal or an RC network for setting the system's operating frequency). It also gives the chip designer maximum freedom in the structure of the internal clocking scheme, because there is no need for high-level, accurate clock inputs.

Through efficient use of internal bus transfers, most instructions can be executed in a single-cycle length. The exceptions are those instructions which require a second memory fetch or an external I/O transfer. In these cases, only a second cycle is required. Moreover, limiting instructions to two lengths reduces the complexity of the internal state generator. Since 70% of all instructions are executed in a single cycle, program-execution times and program-storage size are still minimized.

The multiplexed bus for address and data during external memory references maximizes the number of I/O pins available on a cost-effective 40-pin dual in-line package. For external program-memory references, bits of an additional I/O port are used for address lines, with the input/output data being restored after the memory



5. Latching on. Adding standard memories to the system is quickly done with external latch 8212, which allows standard memory parts to be hooked directly onto the 8748/8048 bus. Operation of the latch is under the control of signals from the processor.



6. Mixing it up. Besides the main system port, 0, the processor chip has two others, 1 and 2, which allow inputs and outputs to be mixed on the same port. Here, writing a 0 causes the pull-down devices to sink the TTL load; writing a 1 calls on the 50-kilohm pull-up resistor.

reference is finished with the address.

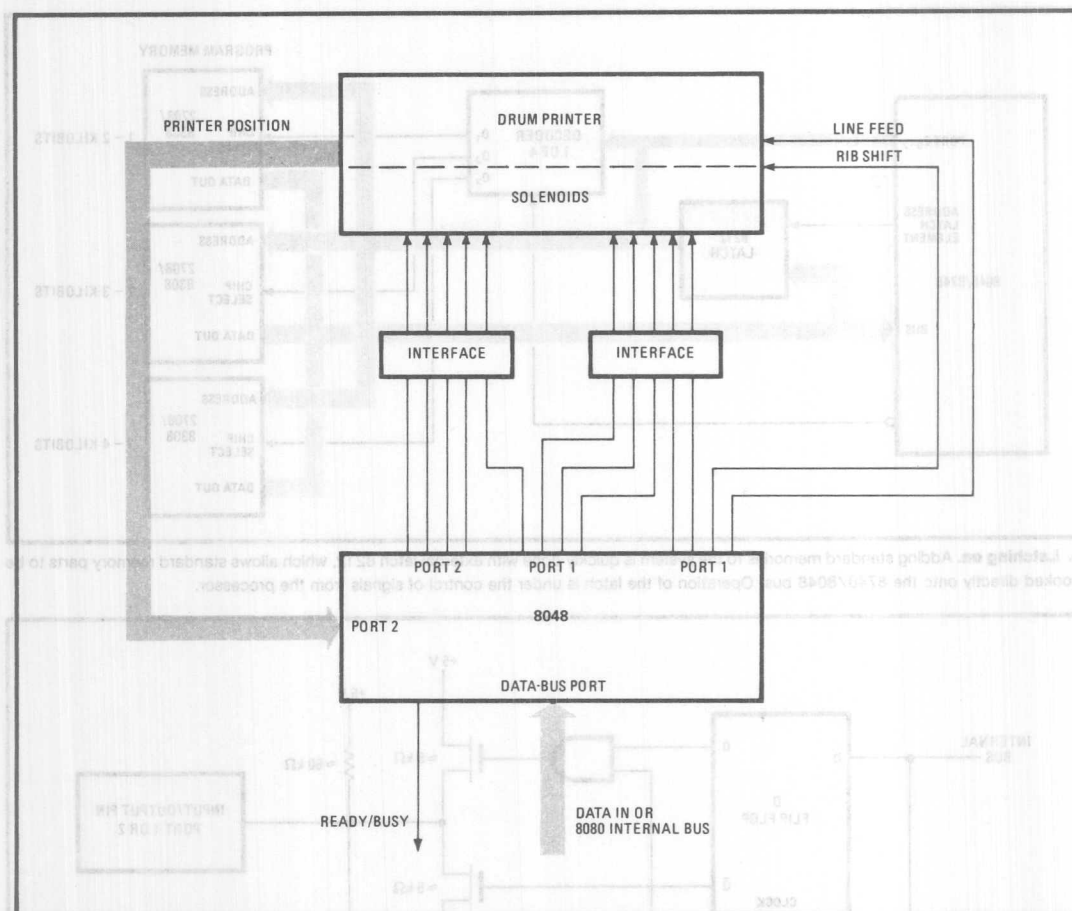
One key to the simple operation of the 8748/8048 chip is the straightforward program sequences and timing needed for executing an instruction cycle (Fig. 3). Each cycle consists of five states. Instruction input is made in state 1, and decoding and pc incrementing is made in state 2. State 3 starts the beginning of the program execution, which can run through states 4 and 5. Simultaneously, the next cycle's program address is made in state 3, a pipelining (paralleling) of operations that increases device throughput significantly.

Because the chip is built with depletion-load silicon-gate n-channel technology, it operates off a single 5-volt supply with inputs and outputs that are compatible with

both transistor-transistor-logic and complementary metal-oxide-semiconductor devices. Instruction cycle time is a modest 2.5 microseconds and power consumption is a low 400 mW. Depletion-load techniques also pay off in practical chip sizes for volume production; the 8048 also is slightly over 200 mils on a side, while the 8748, with its big 8-k EPROM, is 221 by 261 mils.

Storing data in the scratchpad is simple, because part of the RAM can be reserved for one or two banks of 8-bit working registers—eight registers per bank (Fig. 4). The scratchpad also contains the subroutine address stack, which can be eight levels deep. Each location can accommodate the 12-bit pc and 4-bit status data.

Since all locations in the stack are indirectly address-



7. Going it alone. This one-chip scale controller is made possible by the extensive I/O capability of the 8748 processor, which can accommodate a 24-key keyboard and all the interfacing needed to control 14 seven-segment LED arrays, including a decimal point.

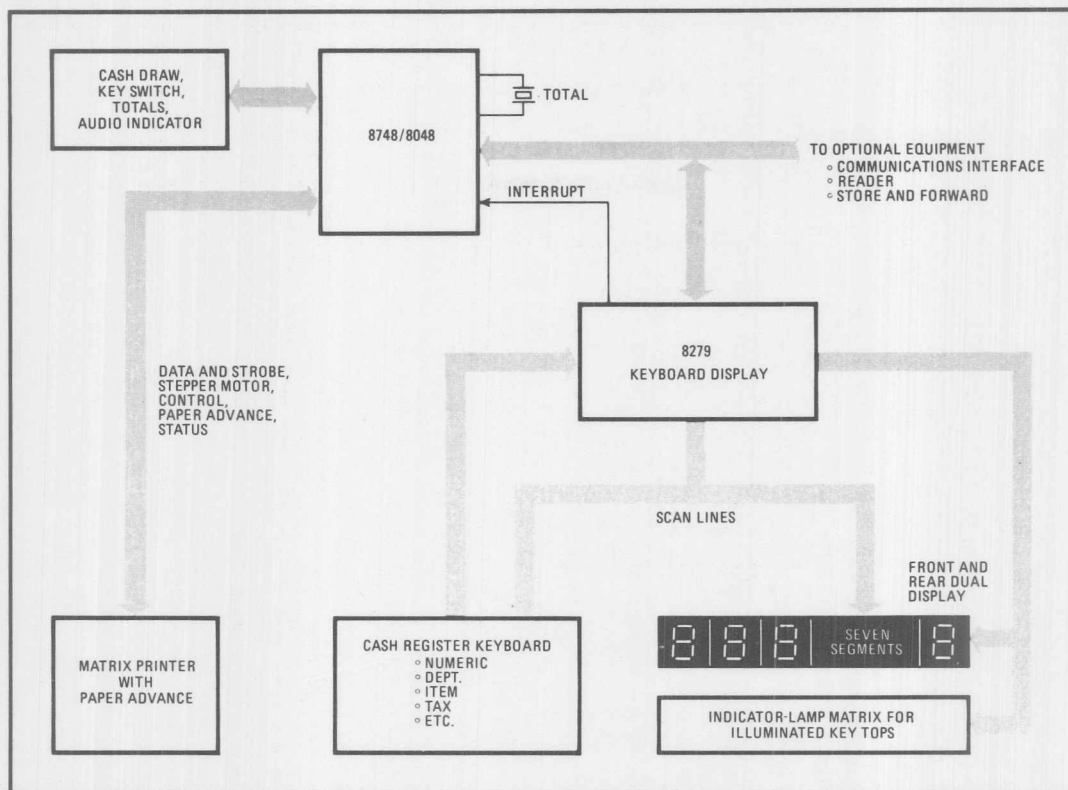
able, the second register bank and any portion of the stack may be used for data memory as well. This gives the user an option of having the data memory be 32 by 8 bits if all the stack and both register banks are used for program counting and status data, or 56 by 8 bits if only one register bank is used.

Program memory

The resident program memories on the 8048/8748 chips are handled so that they can be operated alone for programs of 1,024 bytes or less or combined with external ROM for expanded systems requiring larger programs. The program counter that feeds the memory is split into two parts. The low-order 8 bits can either address the resident 1-k ROM or be routed externally when addressing beyond 1,024 bits. (Since the 8035 contains no internal ROM, all address fetches are external.) The upper 4 bits of the program counter, located near port 2 (see Fig. 2), are gated out on that port for external reference. Two of these most significant 4 bits are then used for internal addressing requirements.

There are two ways to expand program memory of the MCS-48 family. The special parts such as the 8755 2-k-by-8 EPROM or 8355 2-k-by-8 ROM may be used. Besides I/O lines, they also contain appropriate buffers to demultiplex the 8-bit bus from the microcomputer chips to receive address and send back program-memory instructions. Alternately, standard memory parts, such as a 2708 EPROM or 8308 ROM may be used (Fig. 5). An external latch, such as the 8212, would latch up the address from the bus (via a signal from the 8048 or 8748) so that data could be returned on the bus. The high-order 4 bits of the address do not have to be latched, since they are not on the multiplexed bus.

The ALU, in conjunction with the accumulator, provides a full array of binary-and-decimal arithmetic, logic, shift, and increment/decrement functions. For example, the accumulator may be exchanged between registers, data memory, and program memory. Both the timer/counter and the program-status word are also accessible to the accumulator, through a latch that facilitates the accumulator source/destination instruc-



8. Working together. Proof of the MCS-48's ability to handle large systems is this gas-pump controller. The 8243 I/O expander chips allow the processor to interface with 47 lines and a USART communicating with a central control unit inside the service station.

tions. Here, the ALU generates a carry output fully accessible to the programmer under program control.

The timer/event counter is an 8-bit register that can operate in one of two modes, selectable under software control. As a timer, the device measures elapsed time. It is fed by the crystal frequency, divided by 280. At maximum frequency, the result is about 80 μ s per increment, or about 20 milliseconds over the counter range. As an event counter, a test line is designated to count 0 to 1 transitions of external events. As many as 256 transitions may be accommodated.

Both the timer and the counter indicate overflow by a maskable internal interrupt or by a testable flag bit. The internal interrupt may also be used to provide the system with a second external interrupt.

The input/output facilities of the 8048/8748 have been designed for maximum flexibility and expansion and are fully TTL-compatible. The basic facilities consist of three 8-bit I/O ports plus three test/interrupt inputs.

Port 0, called the bus, provides for system expansion. In essence, the port makes the bus completely compatible with an 8080 bus, so that all 8080 peripherals can be used with the MCS-48 family. In conjunction with four control and strobe lines, the port may be used for bidirectional interfacing to memories and I/O elements. For free-standing operations, it may be statically latched

or used as a general input port.

The remaining two I/O ports, 1 and 2, are termed "quasi-bidirectional" (Fig. 6). They allow inputs and outputs to be mixed on the same port. When writing a 0 (low value) to these ports, the pull-down device sinks the TTL load. When writing a 1, a large current is supplied through both pull-up devices to allow a fast transition. After a short time, they shut off and the pull-up of the 50-kilohm resistor sustains the 1 level.

Applying the 8748/8048

Two applications show the range of complexity that can be accommodated with this family. Figure 7 shows a typical minimum-chip MCS-48 system, in this case, a drum printer controller. The three output ports allow the one-chip 8048 to control the printer position, ribbon shift, and line feed. Two interface drivers operate the solenoids.

Figure 8 shows a far more complex system, in which the MCS-48 implements a low-cost point-of-sale terminal. The I/O capability of the 8748/8048 chip can be expanded to control and monitor many cash-register operations. These might include cash in the drawer, key switch, totals, audio indicator, as well as matrix printer, cash-register keyboard, seven-segment display, and a variety of optional equipment. □

2-3	Introduction
2-3	The MCS-48™ Family
2-5	Analog I/O
2-9	Table Lookup Techniques
2-10	Receiving Serial Codes— Basic Approaches
2-14	Receiving Serial Codes— A More Sophisticated Algorithm
2-24	Transmitting Serial Code
2-24	Generating Parity
2-25	Conclusion

August 1977

Application Techniques for the MCS-48™ Family

Lionel Smith
Microcomputer Applications

Application Techniques for the MCS-48™ Family

Contents

Introduction	2-3
The MCS-48™ Family	2-3
Analog I/O	2-5
Table Lookup Techniques	2-9
Receiving Serial Codes— Basic Approaches	2-10
Receiving Serial Code— A More Sophisticated Algorithm	2-14
Transmitting Serial Code	2-24
Generating Parity	2-24
Conclusion	2-25

INTRODUCTION

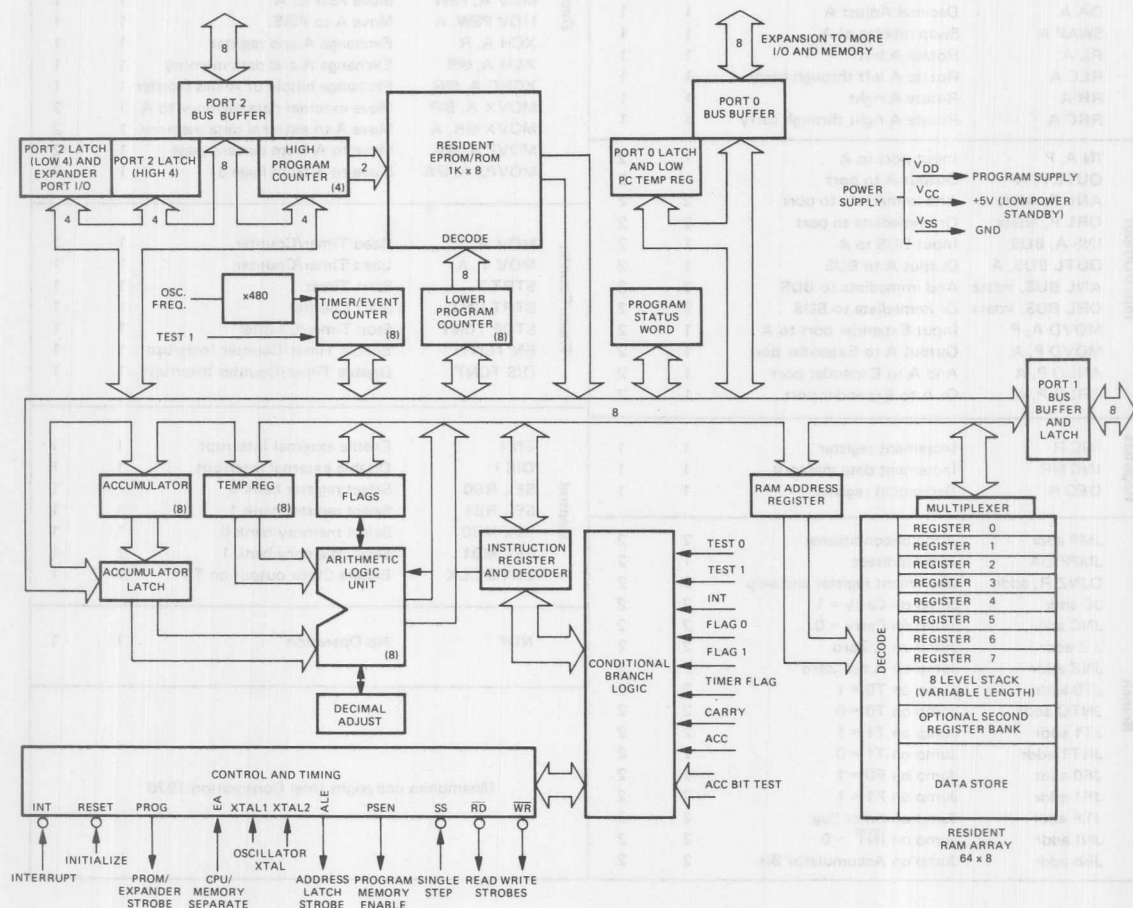
The INTEL® MCS-48™ family consists of a series of seven parts, including three processors, which take advantage of the latest advances in silicon technology to provide the system designer with an effective solution to a wide variety of design problems. The significant contribution of the MCS-48 family is that instead of consisting of integrated microcomputer components it consists of integrated microcomputer systems. A single integrated circuit contains the processor, RAM, ROM (or PROM), a timer, and I/O.

This application note suggests a variety of application techniques which are useful with the MCS-48. Rather than presenting the design of a complete system it describes the implementation of "subsystems" which are common to many micropro-

cessor based systems. The subsystems described are analog input and output, the use of tables for function evaluation, receiving serial code, transmitting serial code, and parity generation. After an overview of the MCS-48 family these areas are discussed in a more or less independent manner.

THE MCS-48™ FAMILY

The processors in the MCS-48 family all share an identical architecture. The only significant difference is the type of on board program storage which is provided. The 8748 (see Figure 1) includes 1024 bytes of erasable, programmable, ROM (EPROM), the 8048 replaces the EPROM with an equivalent amount of mask programmed ROM, and the 8035 provides the CPU function with no on board program storage. All three of these processors



MCS-48™ Internal Structure

INSTRUCTION SET

	Mnemonic	Description	Bytes	Cycle		Mnemonic	Description	Bytes	Cycles
Accumulator	ADD A, R	Add register to A	1	1	Subroutine	CALL	Jump to subroutine	2	2
	ADD A, @R	Add data memory to A	1	1		RET	Return	1	2
	ADD A, #data	Add immediate to A	2	2		RETR	Return and restore status	1	2
	ADDC A, R	Add register with carry	1	1	Flags	CLR C	Clear Carry	1	1
	ADDC A, @R	Add data memory with carry	1	1		CPL C	Complement Carry	1	1
	ADDC A, #data	Add immediate with carry	2	2		CLR F0	Clear Flag 0	1	1
	ANL A, R	And register to A	1	1		CPL F0	Complement Flag 0	1	1
	ANL A, @R	And data memory to A	1	1		CLR F1	Clear Flag 1	1	1
	ANL A, #data	And immediate to A	2	2		CPL F1	Complement Flag 1	1	1
	ORL A, R	Or register to A	1	1	Data Movers	MOV A, R	Move register to A	1	1
	ORL A, @R	Or data memory to A	1	1		MOV A, @R	Move data memory to A	1	1
	ORL A, #data	Or immediate to A	2	2		MOV A, #data	Move immediate to A	2	2
	XRL A, R	Exclusive Or register to A	1	1		MOV R, A	Move A to register	1	1
	XRL A, @R	Exclusive or data memory to A	1	1		MOV @R, A	Move A to data memory	1	1
	XRL A, #data	Exclusive or immediate to A	2	2		MOV R, #data	Move immediate to register	2	2
	INC A	Increment A	1	1		MOV @R, #data	Move immediate to data memory	2	2
	DEC A	Decrement A	1	1		MOV A, PSW	Move PSW to A	1	1
	CLR A	Clear A	1	1		MOV PSW, A	Move A to PSW	1	1
	CPL A	Complement A	1	1		XCH A, R	Exchange A and register	1	1
	DA A	Decimal Adjust A	1	1		XCH A, @R	Exchange A and data memory	1	1
	SWAP A	Swap nibbles of A	1	1		XCHD A, @R	Exchange nibble of A and register	1	1
	RL A	Rotate A left	1	1		MOVX A, @R	Move external data memory to A	1	2
	RLC A	Rotate A left through carry	1	1		MOVX @R, A	Move A to external data memory	1	2
	RR A	Rotate A right	1	1		MOVP A, @A	Move to A from current page	1	2
	RRC A	Rotate A right through carry	1	1		MOVP3 A, @A	Move to A from Page 3	1	2
Input/Output	IN A, P	Input port to A	1	2	Timer/Counter	MOV A, T	Read Timer/Counter	1	1
	OUTL P, A	Output A to port	1	2		MOV T, A	Load Timer/Counter	1	1
	ANL P, #data	And immediate to port	2	2		STRT T	Start Timer	1	1
	ORL P, #data	Or immediate to port	2	2		STRT CNT	Start Counter	1	1
	INS A, BUS	Input BUS to A	1	2		STOP TCNT	Stop Timer/Counter	1	1
	OUTL BUS, A	Output A to BUS	1	2		EN TCNTI	Enable Timer/Counter Interrupt	1	1
	ANL BUS, #data	And immediate to BUS	2	2		DIS TCNTI	Disable Timer/Counter Interrupt	1	1
	ORL BUS, #data	Or immediate to BUS	2	2	Control	EN I	Enable external interrupt	1	1
	MOVD A, P	Input Expander port to A	1	2		DIS I	Disable external interrupt	1	1
	MOVD P, A	Output A to Expander port	1	2		SEL RB0	Select register bank 0	1	1
Branch	ANLD P, A	And A to Expander port	1	2		SEL RB1	Select register bank 1	1	1
	ORLD P, A	Or A to Expander port	1	2		SEL MB0	Select memory bank 0	1	1
						SEL MB1	Select memory bank 1	1	1
	JMP addr	Jump unconditional	2	2		ENTO CLK	Enable Clock output on T0	1	1
	JMPP @A	Jump indirect	1	2	NOP		No Operation	1	1
	DJNZ R, addr	Decrement register and skip	2	2					
	JC addr	Jump on Carry = 1	2	2	Mnemonics copyright Intel Corporation 1976				
	JNC addr	Jump on Carry = 0	2	2					
	J Z addr	Jump on A Zero	2	2					
	JNZ addr	Jump on A not Zero	2	2					
	JT0 addr	Jump on T0 = 1	2	2					
	JNT0 addr	Jump on T0 = 0	2	2					
	JT1 addr	Jump on T1 = 1	2	2					
	JNT1 addr	Jump on T1 = 0	2	2					
	JF0 addr	Jump on F0 = 1	2	2					
	JF1 addr	Jump on F1 = 1	2	2					
	JTF addr	Jump on timer flag	2	2					
	JNI addr	Jump on INT = 0	2	2					
	JBb addr	Jump on Accumulator Bit	2	2					

Figure 2. 8048/8748/8035 Instruction Set

operate from a single 5-volt power supply. The 8748 requires an additional 25-volt supply only while the on board EPROM is being programmed. When installed in a system only the 5-volt supply is needed. Aside from program storage, these chips include 64 bytes of data storage (RAM), an eight bit timer which can also be used to count external events, 27 programmable I/O pins and the processor itself. The processor offers a wide range of instruction capability including many designed for bit, nibble, and byte manipulation. The instruction set is summarized in Figure 2.

Aside from the processors, the MCS-48 family includes 4 devices: one pure I/O device and 3 combination memory and I/O devices. The pure I/O device is the 8243, a device which is connected to a special 4 bit bus provided by the MCS-48 processors and which provides 16 I/O pins which can be programmatically controlled.

The combination memory and I/O devices consist of the 8355, the 8755, and the 8155. The 8355 and the 8755 both provide 2,048 bytes of program storage and two eight bit data ports. The only difference between these devices is that the 8355 contains masked program ROM and the 8755 contains EPROM. The 8155 combines 256 bytes of data storage (RAM), two eight bit data ports, a six bit control port, and a 14 bit programmable timer.

Figure 3 shows the various system configurations which can be achieved using the MCS-48 family of parts. It should also be noted that eight of the processors' I/O lines have been configured as a bidirectional bus which can be used to interface to standard Intel peripheral parts such as the 8251 USART (for serial I/O), the 8255A PPI (provides 24 I/O lines) and the complete range of memory components.

More detailed information concerning the MCS-48 family can be obtained from the "MCS-48 Microcomputer User's Manual" which provides a complete description of the MCS-48 family and its members. A general familiarity with this document will make the application techniques which follow easier to understand.

ANALOG I/O

If analog I/O is required for a MCS-48™ system there are many alternatives available from the makers of analog I/O modules. By searching through their catalogs it is possible to find almost any combination of features which is technically feasible. Perhaps the best example of such modules are the MP-10 and MP-20 hybrid modules recently introduced by Burr-Brown Research Corporation. The MP-10 provides two analog outputs and the MP-20 provides 16 analog inputs. Both of these units were

		[] Number of Available Timers		() Number of Available I/O Lines	
DATA MEMORY (RAM)	1088				
	1K	8048 4-8155 [5] (101)	8035 8355 4-8155 [5] (116)	8048 8355 4-8155 [5] (116)	8035 2-8355 4-8155 [5] (131)
	832				
	768	8048 3-8155 [4] (80)	8035 8355 3-8155 [4] (95)	8048 8355 3-8155 [4] (95)	8035 2-8355 3-8155 [4] (110)
	578				
	512	8048 2-8155 [3] (59)	8035 8355 2-8155 [3] (74)	8048 8355 2-8155 [3] (74)	8035 2-8355 2-8155 [3] (89)
	320				
	256	8048 8155 [2] (38)	8035 8355 8155 [2] (53)	8048 8355 8155 [2] (53)	8035 2-8355 8155 [2] (68)
	64	8048 [1] (24)	8035 8355 [1] (28)	8048 8355 [1] (28)	8035 2-8355 [1] (43)
		1K	2K	3K	4K
		PROGRAM MEMORY (ROM)			

Figure 3. The Expanded MCS-48™ System

specifically designed to interface with microprocessors.

A block diagram of the MP-10 is shown in Figure 4. It consists of two eight bit digital to analog converters, two eight bit latches which are loaded from the data bus, and address decoding logic to determine when the latches should be loaded. The D/A converters each generate an analog output in the range of 10 volts with an output impedance of 1Ω. Accuracy is ±0.4% of full scale and the output is stable 25μsec after the eight bit binary data is loaded into the appropriate latch. The latches are loaded by the write pulse (WR) whenever the proper address is presented to the MP-10. The lower two addresses (A₀ and A₁) are used internally by the device. Addresses A₂ & A₃ are compared with the address determination inputs B₂ and B₃. If their signals are found to be equal, and if addresses A₄-A₁₃ are all high, then the device is selected and one of the latches will be loaded. Address bit A₁ selects between output 1 and output 2. If address bit A₀ is set then the initialization channel of the DIA is selected. In order to prepare for operation a data pattern of 80H must

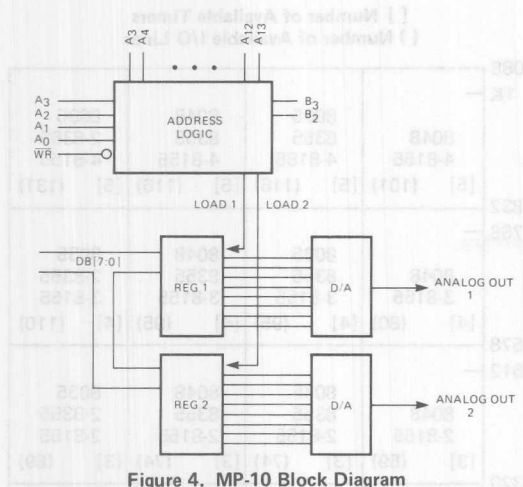


Figure 4. MP-10 Block Diagram

be output to this channel following the reset of the device.

A block diagram of the MP-20 analog to digital converter is shown in figure 5. This unit consists of a 16 input analog multiplexer, an instrumentation amplifier, an eight bit successive approximation analog to digital converter, and control logic. The 16 input multiplexer can be used to input either 16 single ended or 8 differential inputs. The output from the multiplexer is fed into the instrumentation amplifier which is configured so that it can easily be strapped for single ended 0-5 volt inputs, single ended ± 5 volt inputs, or differential 0-5 volt signals. Provisions are made for an external gain control resistor on the amplifier. The gain control equation is:

$$G = 2 + \frac{50k\Omega}{R_{ext}}$$

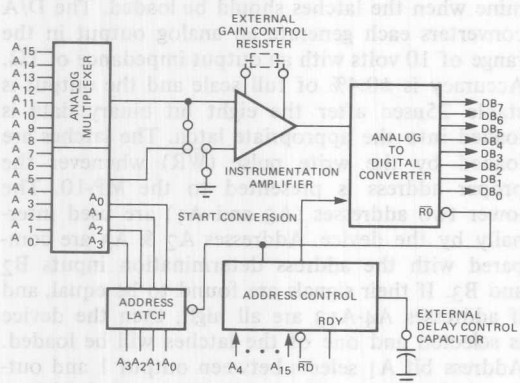


Figure 5. MP-20 Analog Subsystem

With no R_{ext} ($R_{ext} = \infty$) the gain is two and the input is 0-5 or ± 5 volts full scale. Adding an external resistor results in higher gain so that low level (± 50 mV) signals from thermocouples and strain gauges can be accommodated. The output from the amplifier is applied to the actual A/D converter which provides an eight bit output with guaranteed monotonicity and an accuracy of $\pm 0.4\%$ of full scale. Note that this accuracy is specified for the entire module, not just for the converter itself. The control logic monitors address lines A15 through A4 to determine when the address of the unit has been selected. An address that the unit will respond to is determined by 11 address control pins, labeled $\overline{A4}$ through $\overline{A14}$. If one of these pins is tied to a logic 0 then the corresponding address pin must be high in order for the unit to be selected. If the pin is tied to a logic 1 then the corresponding address pin must be low. If the address of the module is selected when \overline{MEMR} pulse occurs, the lower four addresses ($A3-A0$) are stored in a latch which addresses the multiplexer. The coincidence of the proper address and \overline{MEMR} also initiates a conversion and gates the output of the converter on to the eight bit data bus.

The control logic of the MP-20 was designed to operate directly with an MCS-80TM system. When a \overline{MEMR} occurs and a conversion is initiated the MP-20 generates a READY signal which is used to extend the cycle of the 8080A for the duration of the conversion. READY is brought high after the conversion is complete which allows the 8080A to initiate a conversion and read the resulting data in a single, albeit long, memory or I/O cycle. The conversion time of the MP-20 depends on the gain selected for the amplifier. With no external resistor ($R = \infty$) the gain is two and the conversion time is 35 μ sec. For $R = 510\Omega$ the gain is:

$$G = 2 + \frac{50k\Omega}{.51k\Omega} \approx 100$$

and the conversion time becomes 100 μ sec. These settling times are specified in the MP-20 data sheet and range from 35 to 175 microseconds. The READY timing is controlled by an external capacitor. For a gain of 2 no external capacitor is required but if higher gains are selected a capacitor is needed to extend the timing.

A schematic showing both the MP-10 D/A and the MP-20 A/D connected to the 8748 is shown in Figure 6. This configuration, which consists of only four major components, gives an excellent example of what modern technology can do for

[illegible]

Pin	Function	Pin	Function
1	DBIN	21	A_4
2	OUTPUT SELECT	22	A_5
3		23	A_6
4	COMP IN	24	A_7
5	BPO	25	A_8
6	R1	26	A_9
7	IA OUT	27	A_{10}
8	MP20	28	A_{11}
9	IA IN LO	29	A_{12}
10	IA IN HI	30	A_{13}
11	MUX OUT HI	31	A_{14}
12	MUX OUT LO	32	A_{15}
13	MUX ENAB2	33	A_{16}
14	SIN/DIFF	34	A_{17}
15	MEMR	35	A_{18}
16	D_0	36	A_{19}
17	D_1	37	A_{20}
18	D_2	38	A_{21}
19	D_3	39	A_{22}
20	D_4	40	A_{23}
	D_5		A_{24}
	D_6		A_{25}
	D_7		A_{26}
	RST		A_{27}
	O_7		A_{28}
			A_{29}
			A_{30}
			A_{31}
			A_{32}
			A_{33}
			A_{34}
			A_{35}
			A_{36}
			A_{37}
			A_{38}
			A_{39}
			A_{40}
			A_{41}
			A_{42}
			A_{43}
			A_{44}
			A_{45}
			A_{46}
			A_{47}
			A_{48}
			A_{49}
			A_{50}
			A_{51}
			A_{52}
			A_{53}
			A_{54}
			A_{55}
			A_{56}
			A_{57}
			A_{58}
			A_{59}
			A_{60}
			A_{61}
			A_{62}
			A_{63}
			A_{64}
			A_{65}
			A_{66}
			A_{67}
			A_{68}
			A_{69}
			A_{70}
			A_{71}
			A_{72}
			A_{73}
			A_{74}
			A_{75}
			A_{76}
			A_{77}
			A_{78}
			A_{79}
			A_{80}
			A_{81}
			A_{82}
			A_{83}
			A_{84}
			A_{85}
			A_{86}
			A_{87}
			A_{88}
			A_{89}
			A_{90}
			A_{91}
			A_{92}
			A_{93}
			A_{94}
			A_{95}
			A_{96}
			A_{97}
			A_{98}
			A_{99}
			A_{100}
			A_{101}
			A_{102}
			A_{103}
			A_{104}
			A_{105}
			A_{106}
			A_{107}
			A_{108}
			A_{109}
	</		

5
4 D₆

[illegible]

the system designer. The four components provide:

- An eight bit microprocessor
- 64 bytes of RAM
- 1024 bytes of UV erasable PROM
- A timer/event counter
- 16 digital I/O pins
- 2 testable input pins
- An interrupt capability
- 16 eight bit analog inputs
- 2 eight bit analog outputs

The MCS-48 communicates with the D/A and A/D converters in a memory mapped mode (i.e., it treats the devices as if they were external RAM). By setting an address in either R0 or R1 and then executing a MOVX the software can transfer data between the accumulator and the analog I/O. When the MCS-48 executes the MOVX instruction it first sends the eight bit address out on the bus and strobes it into the 8212 latch with the ALE (Address Latch Enable) signal. After the address is latched, the MCS-48 uses the same bus to transfer data to or from the accumulator. If data is being sent out (MOVX ∂R_j , A) the \overline{WR} strobe is used; if the data is being moved into the accumulator (MOVX A, ∂R_j) the \overline{RD} strobe is used. The one shots on the \overline{WR} line are used to delay the write strobe of the MCS-48 to meet the data set up specifications of the MP-10.

In order to provide reset capability for the analog devices without dedicating an I/O pin from the MCS-48, special addresses are used as reset channels. Executing any MOVX with an address of 0XXXXXXX will reset the A/D module; a similar operation with an address of X1XXXXXX will reset the D/A; a MOVX with an address of 01XXXXXX will reset both devices. All data transfers are accomplished with the upper two bits of the address field equal to 10. A summary of the addressing of the analog devices is shown in Table 1. Notice that except for an initialization channel for the D/A (which must

Table 1. Analog Interface Addresses

INPUT OR OUTPUT		
0 X X X	X X X X	Reset A/D
X 1 X X	X X X X	Reset D/A
INPUT		
0 0 1 1	n n n n	Read A/D Channel n n n n
OUTPUT		
1 0 1 1	0 0 0 1	Initialize D/A
1 0 1 1	0 0 0 0	Write Channel 1
1 0 1 1	0 0 1 0	Write Channel 2

be written to following a reset to initialize its internal logic) all channels involve some form of data transfer.

As was mentioned previously, the MP-20 was designed to use the READY line of the 8080A. Obviously this presents a problem since the MCS-48 does not support a READY line (with its attendant requirement of entering WAIT state). The necessity of a READY input can be overcome by performing a read operation to set the channel address, waiting the required delay (35 μ sec for a gain of two) and then performing a second read to actually obtain the data. The second read will read in the data from the channel selected by the first read irrespective of the channel selected for the second read. Thus it is possible to use the second read to set up the channel for the third read. Each read can read in the current channel and select the next channel for conversion.

The MP-20 is shown in Figure 6 strapped to input 16 single ended ± 5 volts signals. Programs which were used to test this configuration are shown in Figure 7. The first of these programs uses the D/A converter to generate sawtooth waveforms by outputting an incrementing value to the D/A converters. The second program scans the analog inputs and stores their digital values in a table located in RAM.

```

LOC  OBJ      SEQ      SOURCE STATEMENT
0
1
2 : TEST PROGRAM FOR ANALOG OUTPUT
3 : THIS PROGRAM OUTPUTS A SAW-
4 : TOOTH WAVEFORM BY OUTPUTTING
5 : AN INCREMENTING PATTERN.
6
7
8
9 : EQUATES
10
11
12
13 INITCH EQU 0B3H ; D/A INITIALIZATION CHANNEL
14 INITDT EQU 0BH ; D/A INITIALIZATION DATA
15 DATCH EQU 0BH ; D/A DATA CHANNEL
16
17 : START OF TEST
18
19
20 ORG 100H ; INITIALIZE D/A
21
22 START: MOV A, #INITDT
23         MOV R0, #INITCH
24         MOVX @R0, A
25         ; TEST LOOP-OUTPUT SAWTOOTH
26 LOOP:  MOV R0, #DATCH
27         INC A
28         MOVX @R0, A
29         JMP LOOP
30
31 END ; END OF PROGRAM

```

Figure 7a. D/A Exercise Program


```

LOC OBJ  SEQ  SOURCE STATEMENT
#
2
3 : TEST PROGRAM FOR ANALOG INPUT
4 : THIS PROGRAM SCANS THE INPUT CHANNELS
5 : AND STORES THE READINGS IN A TABLE
6 : STARTING AT BUFF.
7
8
9 : -----
10 : EQUATES
11 : -----
12
13 BUFF EQU 2BH : START OF BUFFER
14 MAXCH EQU 15 : NO OF ANALOG INPUTS
15 AINCH EQU 8BH : BASE ADDRESS OF ANALOG INPUTS
16 TICK EQU 5 : EXECUTION TIME OF DJNZ
17
18 : -----
19 : START OF TEST
20 : -----
21
22 ORG 18BH : SETUP TO SCAN ANALOG INPUTS
23 START: MOV R1, #BUFF+MAXCH
24 MOV R3, #MAXCH
25 MOV R8, # (AINCH+MAXCH)
26 : SELECT CHANNEL 15
27 MOVX A, @R8
28 : WAIT >48 MICROSECONDS
29 MOV R4, #48/TICK
30 DJNZ R4, $
31 : NOW SCAN ANALOGS
32 LOOP: DEC R8
33 : GET DATA
34 MOVX A, @R8
35 : MOVE INTO BUFFER
36 MOV @R1, A
37 : DECREMENT BUFFER POINT
38 DEC R1
39 : PAD 28 MICROSEC
40 MOV R4, #28/TICK
41 DJNZ R4, $
42 : LOOP UNTIL DONE
43 DJNZ R3, LOOP
44 : REPEAT TEST FOREVER
45 JMP START
46 : END OF PROGRAM
47 END

```

Figure 7b. A/D Exercise Program

TABLE LOOKUP TECHNIQUES

In the previous section the interface between analog I/O devices and the MCS-48™ was discussed. In many applications involving analog I/O one quickly finds that nature is inherently nonlinear, and the mathematics involved in 'linearizing it' can tax the computational power of the microprocessor, particularly if it has other tasks to perform. Problems of this nature are good candidates for the use of tables.

As an example of how tables can be used as part of an analog output scheme, consider a system which requires an MCS-48 to output a variable frequency sinusoidal waveform. One method of performing this function would be to use the timer to generate an interrupt at a fixed rate of 256 times the desired output frequency. At each interrupt the appropriate value of the sine function could be calculated from the MacLaurin series:

$$\sin x = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} \cdots \frac{(-1)^k x^{2k+1}}{(2K+1)!}$$

Where K is chosen to be large enough to provide the required accuracy.

The above calculation, although conceptually simple, would be time consuming and would severely limit the possible output frequencies which could be obtained. As an alternative to calculating these values in real time, the values could be precalculated off line and stored in a table. Upon each interrupt the MCS-48 would merely have to retrieve the appropriate value from the table and output it to the D/A converter. The MCS-48 provides a special instruction which can be used to access data in a table. If the table is stored in the last 256 bytes of the first kilobyte of MCS-48 memory then the table lookup can be performed by loading the independent variable (time in this case) into the accumulator and executing the instruction.

MOVP3 A, @A.

This instruction uses the initial contents of the accumulator to index into page 3 of program storage. The location pointed to is read and the contents placed in the accumulator. If (as is often the case) a table of fewer than 256 entries is required, then the table can be located in any page of program memory and the instruction:

MOVP A, @A

can be used to retrieve data from the table. This instruction operates in the same manner as does the previous instruction except that the current page of program storage is assumed to contain the table.

If it is possible to devote slightly more of the microprocessor's time to the table look up process, then a much smaller table can often be utilized by taking advantage of interpolation to determine values of the function between values which are actual entries in the table. As an example of this

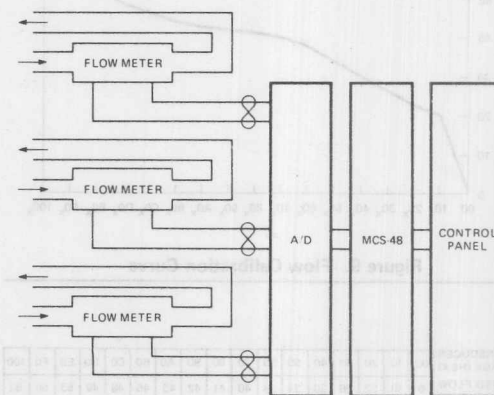


Figure 8. Flow Monitoring System

process consider the hypothetical system shown in Figure 8. The purpose of this system is to measure the flow through the three pipes, add them, and display the total flow on the control panel. The system consists of three flow meters which generate a differential voltage which is some function of flow, an A/D system with at least three differential inputs, an MCS-48, and a control panel. The schematic shown in Figure 6 could easily become part of this system, with the spare digital I/O of the MCS-48 used as an interface to the control panel. The simplicity of this system is clouded by the flow transducers, which are assumed to be not only nonlinear but also to require individual calibration (this is not an unreasonable assumption for a flow transducer). By using a table look up process and an 8748 the flow transducers can be calibrated and the results of the calibration tests stored directly in tables in the 8748. (The 8748 has a PROM in place of the ROM of the 8048 and thus makes such 'one off' programming practical.)

The results which might be obtained from calibrating one of the flow meters is shown in Figure 9. The results are plotted as gals/hour versus the measured voltage generated by the transducer. The voltage is shown in hexadecimal form so that it corresponds directly to the digital output of the analog to digital converter. The flow required to generate seventeen evenly spaced voltages (00H-100H in steps of 10H) has been measured and plotted. This information is shown in tabular form in Figure 10. It is necessary to generate a program which will convert any measured input from 00H to FFH into the flow in units which can be interpreted by a human operator. This can easily be done by simple interpolation.

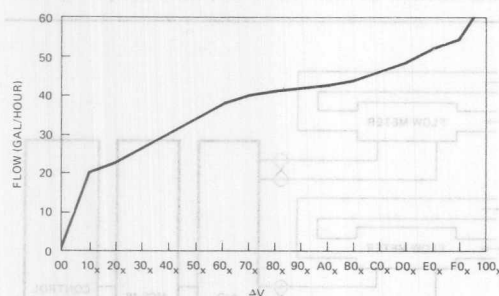


Figure 9. Flow Calibration Curve

TRANSDUCER VOLTAGE (HEX)	00	10	20	30	40	50	60	70	80	90	A0	B0	C0	D0	E0	F0	100
MEASURED FLOW (GAL/HOUR)	0	10	22	26	30	34	38	40	41	42	43	45	48	49	53	56	63

Figure 10. Tabulated Flow Data

The eight bits of independent variable (voltage) can be looked on as two four bit fields. The most significant four bits (7-4) will be used to retrieve one of the table values. The lower four bits (3-0) will be used to interpolate between this value and the value retrieved from the next higher location in the table. If the upper four bits are given the symbol I and the lower four bits the symbol N, then the interpolation can be expressed as:

$$F(x) = F(I) + \frac{N}{16} [F(I+1) - F(I)]$$

Where x is the measured voltage and F(x) is the corresponding flow.

If, as an example, the transducer voltage was measured as 48H then the flow (ref. Figure 10) would be:

$$F = 30 + \frac{8}{16} (34 - 30) = 32$$

A subroutine which implements this calculation is shown in Figure 11. Before it is called the independent variable (V) is placed in the accumulator and register R1 is set to point at the first value in the table. Aside from simple additions and subtractions the only arithmetic required is to multiply two values and then divide them by 16. The multiplication is handled via a subroutine which is also shown in Figure 11. The division by 16 can be performed by a four place right shift followed by a rounding operation. The routine shown will handle a monotonic increasing function of a single independent variable. Fairly simple modifications are required for nonmonotonic functions. Functions of two variables can be handled by interpolating on a plane rather than along a straight line. Although this is more time consuming, requiring an interpolation for each of the independent variables and a third to interpolate the final answer, it still provides a simple means of quickly calculating the required function. The use of tables can offer a powerful technique for function evaluation to the designer.

RECEIVING SERIAL CODE-BASIC APPROACHES

Many microprocessor based systems require some form of serial communication. Serial communication is extensively used because it allows two or more pieces of equipment to exchange information with a minimal number of interconnecting wires. The minimization of interconnecting wires results in simpler, cheaper, interconnects because fewer (or smaller) cables and connectors are required. Since the required number of drivers and receivers required is reduced, it can become economically feasible to provide much higher noise immunity

LOC	OBJ	SEQ	SOURCE STATEMENT	LOC	OBJ	SEQ	SOURCE STATEMENT
		0	*****	011C B3		56	RET
		1				57	
		2	APPROX			58	
		3	AT ENTRY R1 POINTSAT TABLE			59	*****
		4	A HAS INDEPENDENT VARIABLE			60	MULTIPLY
		5				61	*****
		6				62	
		7				63	MULT: MOV COUNT, #8
		8	*****	011D B800		64	MOV AEX, #8
		9	EQUATES			65	
		10	*****			66	LOOPA: CLR C
		11				67	
0000		12	RX0 EQU R0 ; POINTER 0	0122 122B		68	LOOPB: JNB SSUM
0001		13	RX1 EQU R1 ; POINTER1	0124 2A		69	XCH A,AEX
0002		14	AEX EQU R2 ; EXTENSION OF A REGISTER	0125 67		70	RRC A
0003		15	COUNT EQU R3 ; COUNTER	0126 2A		71	XCH A,AEX
0004		16	TEMP EQU R4 ; TEMP STORAGE	0127 67		72	RRC A
		17				73	
		18	*****			74	DJNZ COUNT, LOOPB
		19	APPROXIMATION	012A 03		75	RET
		20	*****			76	
		21		012B 2A		77	SSUM: XCH A,AEX
0100		22	ORG 100H	012C 68		78	ADD A,@RX0
		23		012D 67		79	RRC A
0100 B804		24	APPROX: MOV RX0, #TEMP	012E 2A		80	XCH A,AEX
		25		012F 67		81	RRC A
		26				82	
0102 B800		27	MOV @RX0, #0	0130 B821		83	DJNZ COUNT, LOOPA
0104 30		28	XCHD A,@RX0	0132 B3		84	RET
0105 47		29	SWAP A			85	
		30				86	
0106 69		31	ADD A,RX1			87	*****
0107 A9		32	MOV RX1, A			88	TABLE TO TEST PROGRAM
		33				89	*****
		34				90	
0100 E3		35	MOV P3 A,@A	0300		91	ORG 300H
0109 29		36	XCH A,RX1			92	
010A 17		37	INC A	0300 00		93	TABLE: DB 00
010B E3		38	MOV P3 A,@A	0301 0A		94	DB 10
		39		0302 16		95	DB 22
010C 37		40	CPL A	0303 1A		96	DB 26
010D 69		41	ADD A,RX1	0304 1E		97	DB 30
010E 37		42	CPL A	0305 22		98	DB 34
		43		0306 26		99	DB 38
010F 341D		44	CALL MULT	0307 2B		100	DB 40
0111 B802		45	MOV RX0, #AEX	0308 29		101	DB 41
0113 30		46	XCHD A,@RX0	0309 2A		102	DB 42
0114 47		47	SWAP A	030A 2B		103	DB 43
0115 2A		48	XCH A,AEX	030B 2D		104	DB 45
0116 7219		49	JB3 ADJUST	030C 30		105	DB 48
0118 2A		50	XCH A,AEX	030D 31		106	DB 49
0119 2A		51	ADJUST: XCH A,AEX	030E 35		107	DB 53
011A 17		52	INC A	030F 39		108	DB 56
		53		0310 3F		109	DB 63
011B 69		54	ADD A,RX1			110	
		55				111	END
			RETURN				

Figure 11. Table Lookup With Interpolation

with more sophisticated (and expensive) line terminators. The final, and usually most persuasive, argument in favor of serial communication is that it may be the only method available to accomplish the job. The obvious example of this is telecommunications where it is necessary to encode parallel information into serial format in order to communicate via the telephone network. The intent of this section is to show how the facilities of the MCS-48™ can be brought to bear on the problem of serial communication.

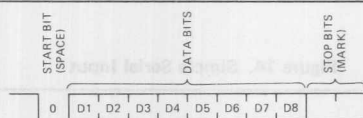


Figure 12. Serial ASCII Code

Probably the most common form of serial communication is that used by the ubiquitous Teletype serial ASCII. This format, shown in Figure 12, consists of a START bit (0 or SPACE) followed by eight data bits which are in turn followed by two STOP bits (1 or MARK). In actual practice the

eighth data bit usually consists of even parity on the remaining seven data bits; for the purposes of this discussion the eighth bit will be considered only as data. A minor variation of this format deletes one of the STOP bits. An algorithm which might be used to sample serial data under software control using a microprocessor is shown in Figure 13. The basic intent of this algorithm is to minimize the effects of distortion and transmission rate variations on the reliability of the communication by sampling each data bit as close to its center as possible. Upon entry to this routine the software first samples the incoming data in a tight loop until it is sensed as a MARK (logical one). As soon as a MARK is detected, a second loop is entered during which the software waits until the received data goes to a SPACE (logical zero). The purpose of this construction is to detect as accurately as possible the leading edge of the START bit. This instant of time will be used as a reference point for sampling all of the following bits in the character. After sensing the leading edge of the START bit a wait of one half the expected bit time is implemented. The period of the incoming signal is called P for convenience. At the end of this wait the serial line is tested—if it is MARK then the START bit was

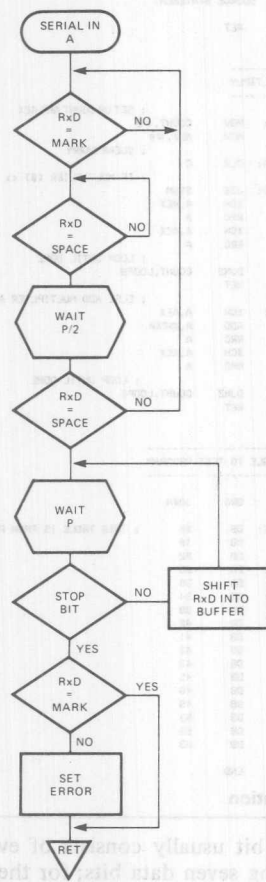


Figure 13. Sample Serial Input Routine

invalid and the process is reinitialized. If the line is still a SPACE, then the START bit is assumed to be valid and a delay of one bit time is started. At the completion of the delay the first data bit is sampled and a new delay of one bit time is initiated. This process is repeated until all eight data bits have been sampled. The last bit sampled is checked to determine if it is a valid STOP bit (a MARK). If it is, the character is assumed to be valid; if it is not, the character has a framing error and is probably invalid. A listing of a program which implements the above procedure is shown in Figure 14.

A disadvantage of the approach outlined in Figure 13 is that while the processor is inputting data serially it must totally dedicate itself to this task. Accurate timing can only be maintained if the program remains in a tight wait loop without allowing itself to be diverted to other functions. During reception of a character from a Teletype

the processor will spend only a 100µsecs or so processing data and the rest of the 100 milliseconds waiting to do the processing at the right time. This lack of efficiency (approximately 0.1%) in the utilization of processing power is why devices such as the 8251 USART find broad application in micro-processor systems.

LOC	OBJ	SEQ	SOURCE STATEMENT
0		0	*****
1		1	
2		2	SIMPLE SERIAL INPUT
3		3	-THIS CODE ASSUMES RxD IS
4		4	CONNECTED TO PIN T8
5		5	
6		6	*****
7		7	
8		8	-----
9		9	EQUATES
10		10	-----
11		11	
0002		12	COUNT EQU R2 ; COUNTER
0008		13	BITNO EQU 0 ; NO OF BITS TO RECEIVE
0002		14	DLYHI EQU 2 ; HI DLY COUNT
00A4		15	DLYLO EQU 0A4H ; LO DLY COUNT
16		16	
0100		17	ORG 100H
0100 2600		18	
0102 3602		19	SERIN: JNT8 \$; LOOP UNTIL RxD=MARK
0104 341C		20	
0106 3600		21	JT8 \$; NOW LOOP UNTIL RxD=SPACE
0108 BA09		22	CALL HBIT ; WAIT 1/2 BIT TIME
010A 341C		23	
010C 341C		24	CALL HBIT ; IF FALSE START REINITIALIZE
		25	
		26	JT8 SERIN ; ELSE SET BIT COUNT
		27	MOV COUNT, #BITNO+1
		28	
		29	LOOP: CALL HBIT ; WAIT 1 BIT TIME
		30	CALL HBIT
		31	
		32	; DECREMENT COUNT
		33	; -IF ZERO EXIT WITH CARRY SET ON
		34	; -FRAMING ERROR
010E EA15		35	DJNZ COUNT, LOAD
0110 97		36	CLR C
0111 3614		37	JT8 EXIT
0113 A7		38	CPL C
0114 83		39	RET ; LOAD DATA
		40	
0115 97		41	LOAD: CLR C
0116 2619		42	JNT8 LLLA
0118 A7		43	CPL C
0119 67		44	RLA ; AND LOOP
011A 24BA		45	JMP LOOP
		46	
		47	-----
		48	DELAY ONE HALF BIT TIME
		49	-----
		50	
		51	; SET UP LOOP
011C BC02		52	HBIT: MOV R4, #DLYHI
		53	; LOOP UNTIL TIME DONE
011E BBA4		54	HLOOP: MOV R3, #DLYLO
0120 EB20		55	DJNZ R3, \$
0122 EC1E		56	DJNZ R4, HLOOP
0124 83		57	RET
		58	; END OF PROGRAM
		59	END

Figure 14. Simple Serial Input

The 8251 USART is simple to interface to the MSC-48. Figure 15 shows such an interface. The USART requires a high speed clock (CLK), an initialization signal (RESET), data clocks (TxC and RxC), and data in order to operate. A circuit showing the connection of an 8748 to an 8251 USART is shown in Figure 15. In the circuit shown the high speed clock (which is used for internal sequencing by the USART) is provided by con-

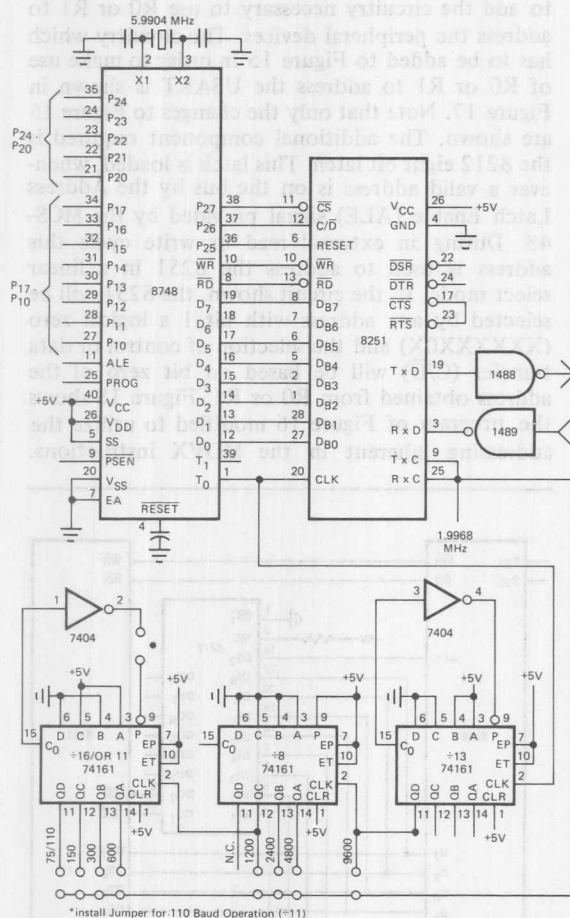


Figure 15. MCS-48™ to 8251 Interface

necting the CLK signal of the USART to the T₀ pin of the MCS-48. The T₀ pin of the MCS-48 can either be used as a directly testable input pin or it can become, under program control, an output pin which oscillates at one third of the crystal frequency. (Note that once this pin is designated by the software to be an output it will remain so until the system is reset.) In Figure 15 the crystal frequency is 5.9904 MHz so the clock provided to the 8251 is 1.9968 MHz, which conforms to its specifications.

The initialization signal to the USART (RESET) is provided programmatically by manipulation of bit 5 of port 2. It was necessary to place the reset of the 8251 under program control for two reasons. The first reason is that the MCS-48 does not supply a reset signal to other devices. The reason for this is that it was felt to be more useful to provide another pin of I/O function instead of a RESET OUT signal

from the MCS-48. Although this situation could have been circumvented by the use of an externally generated reset which drove both the MCS-48 and the 8251, the second reason for program control of the reset to the USART still stands. The USART requires the presence of the CLK signal during reset in order to properly initialize itself. The ENT0 CLK instruction which the MCS-48 must execute before the 8251 will receive the CLK can obviously not be executed until after the system reset has ended. Reset of the USART can be accomplished by the following code segment:

```

    ENT0    CLK            ; TURN ON CLOCK
    ORL     P2, #00100000B ; START RESET
    MOV     R2, #DELAY     ; DELAY USART
LOOP: DJNZ  R2, LOOP       ; RESET TIME
    ANL     P2, #11011111B ; END RESET

```

This code first enables the clock, then asserts the reset signal of a time period determined by the constant DELAY. The delay invoked is $(10 + 5 \cdot \text{DELAY})$ microseconds for $\text{DELAY} > 0$. The USART requires a reset of approximately 6 CLK periods so DELAY is chosen to be 1 which ensures adequate reset timing. Note that for delays this short, NOP instructions could also be used to time the pulse.

The data clocks required by the USART are provided by the modem if the USART is operated in the synchronous mode. In the more common asynchronous mode, however, these clocks must be provided by circuitry associated with the 8251.

The 5.9904 MHz crystal was chosen because the resulting 1.9968 MHz clock to the USART can be evenly divided to provide transmit and receive clocks to the USART. Assuming the USART is in the x16 mode (i.e. it requires data clocks 16 times the baud rate) the 1.9968 MHz signal can be divided by 13 to generate the proper clock rate for 9600 baud operation. This 9600 baud clock can be further divided to give 4800, 2400, 1200, 600, and 300 baud signals. The 1200 baud signal can be divided by 11 to give a 109.1 baud signal which is within 1% of the 110 baud required by Teletypes.

The MCS-48 communicates with the 8251 in a memory mapped mode (i.e. as if the 8251 were external RAM). The instructions available to do this are MOVX @Rj, A which stores the contents of the accumulator at the external RAM location addressed by Rj (j=0 or 1), and its complement, the MOVX A, @ Rj instruction which moves data from the external RAM into the accumulator. Since the MCS-48 multiplexes addresses and data on the same eight bit bus an external latch would be required in order to address the USART with

Figure 16. 8251 Test Program

If more than one 8251 were to be added to the MCS-48, or if other types of peripheral circuitry would be required (e.g. an 8253 timer to generate the data clocks) it would probably become desirable

Refer back to Figure 15 in order to make use of R0 or R1 to address the USART is shown in Figure 17. Note that only the changes to Figure 15 are shown. The additional component required is the 8212 eight bit latch. This latch is loaded, whenever a valid address is on the bus by the Address Latch Enable (ALE) signal provided by the MCS-48. During an external read or write cycle this address is used to address the 8251 in a linear select mode. In the circuit shown, the 8251 will be selected by any address with bit 1 a logical zero (XXXXXX0X) and the selection of control or data transfer (C/D) will be based on bit zero of the address obtained from R0 or R1. Figure 18 shows the program of Figure 16 modified to utilize the addressing inherent in the MOVX instructions.

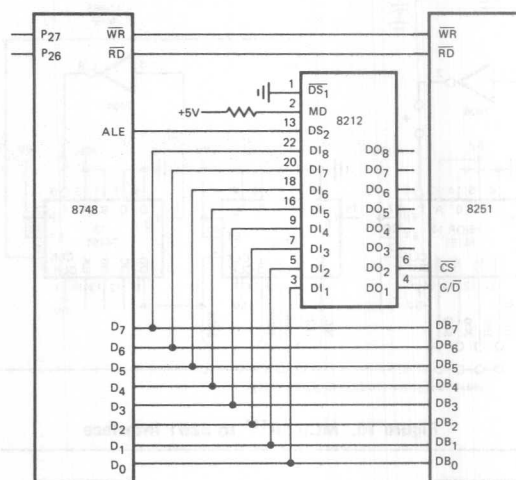


Figure 17. Modified MCS-48 to 8251 Interface

Although the USART does an admirable job of performing the serial I/O function for the MCS-48™, there are some situations where it can not be used. These situations may be caused by economic factors, such as an extremely cost sensitive design, or because the code which must be utilized cannot be accommodated by the USART. An example of such a code will be discussed later. Recall that the principal objection to the approach to serial input shown in Figure 13 was that it consumes much of the processor's power by merely spinning in loops in order to wait preset time delays.

LOC	OBJ	SEQ	SOURCE STATEMENT
0		1	SERIAL TEST
1		2	THIS CODE INITIALIZES THE USART
2		3	AND TRANSMITS AN INCREMENTING
3		4	PATTERN. HARDWARE SHOWN IF FIG 17.
4		5	
5		6	
6		7	
7		8	EQUATES
8		9	
9		10	
0020		11	MCLR EQU 20H ; USART RESET ADDRESS
0001		12	DLY EQU 01H ; USART RESET DELAY
0003		13	UCON EQU 03H ; USART CONTROL ADDRESS
000E		14	MODE EQU 0CEH ; USART MODE
0021		15	CHD EQU 21H ; USART CMD
0003		16	STAT EQU 03H ; USART STATUS
0001		17	VAL EQU R1 ; TEST VALUE
0000		18	DATA EQU 00 ; USART DATA ADDRESS
0100		19	
20		20	DRG 100H
21		21	
22		22	; TURN ON CLOCK
23		23	; AND RESET USART
0100 75		24	TEST: ENT0 CLK
0101 BA20		25	ORL P2, #MCLR
0103 BA01		26	MOV R2, #DLY
0105 EA85		27	LOOP DJNZ R2, LOOP
0107 9ADF		28	ANL P2, #(NOT MCLR)
0109 2303		29	MOV A, #UCON ; SELECT USART CONTROL
010B 230E		30	; SEND MODE AND COMMAND
010D 90		31	MOV A, #MODE
010E 2321		32	MOVX @R0, A ; (CONTENTS OF R0 UNIMPORTANT)
0110 90		33	MOV A, #CHD
0111 90		34	MOVX @R0, A
0112 90		35	
0113 2303		36	; DO FOREVER
0114 67		37	SELECT USART STATUS
0115 E611		38	IF TXRDY=1 THEN
0117 F9		39	DO;
0118 9000		40	OUTPUT VALUE;
011A 90		41	INCREMENT VALUE;
011B 19		42	END;
011C 2411		43	TLP: MOV A, #STAT
52		44	MOVX A, @R0 ; (CONTENTS OF R0 UNIMPORTANT)
53		45	RRC A
		46	JNC TLP
		47	MOV A, VAL
		48	MOV R0, #DATA
		49	MOVX @R0, A
		50	INC VAL
		51	JMP TLP
		52	
		53	END ; END OF PROGRAM

Figure 18. Modified 8251 Test Program

The timer resident on the MCS-48 provides a solution to this problem. Instead of spinning in a loop the program can set the timer for a given interval, start it, and proceed to other tasks. When the timer overflows, an interrupt will be generated to notify the software that the present time period has elapsed. An extension of the algorithm of Figure 13 which uses the timer in this fashion is shown in Figure 19. This algorithm is identical to the preceding one up until the detection of the leading edge of the start bit. At this point the timer is set to one half of the bit time (P) and a return is made to the calling program which can start additional processing. At the completion of this time interval a timer overflow interrupt is generated. When the first interrupt is detected, the serial line is checked to ensure that it is in a spacing condition (valid START bit). If it is, the timer is set to P (to sample the middle of the first data bit) and a return is made to the program which was running when the

All mnemonics copyrighted © Intel Corporation 1976.

interrupt occurred. If the serial line has returned to the MARK state, a status flag is set to indicate an error and a return is made. On subsequent interrupt detection, the data is sampled, the timer is reinitiated, and control is returned to the program which was running when the interrupt occurred. When the last (i.e. STOP) bit is detected a completion flag is set and a return is made to the program running when the timer overflow occurred. By periodically checking the error and completion flags the running program can determine when the interrupt driven receive program has a character assembled for it.

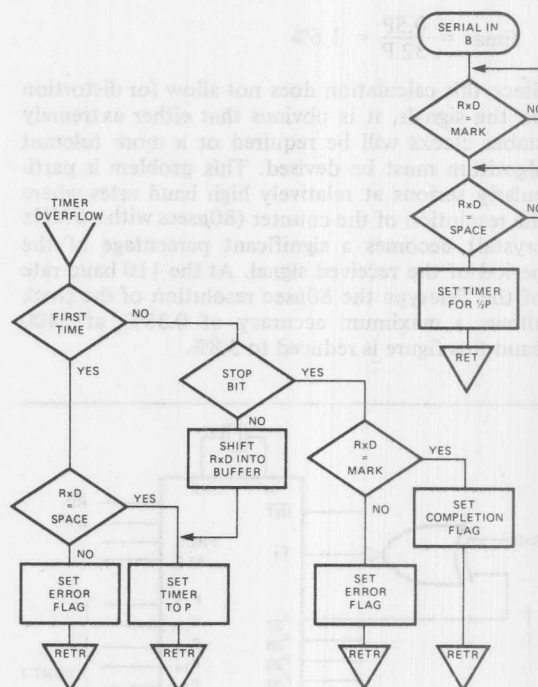


Figure 19. Improved Serial Input Routine

Using the timer to implement time delays as shown in Figure 19 results in considerable savings in processing time; two problems remain, however, which must be solved before an adequate software solution to the problem of receiving serial code can be found. The first problem is that even though the delays between bit samples are implemented via the timer rather than program loops the loop construction is still used to detect the leading edge of

the START bit. Although this results in the waste of processing power, the second problem is even more serious. For longer messages the required accuracy of the clocks becomes more and more stringent. Using the sampling technique discussed a cumulative error of one half a bit time in the time at which a bit sample is taken will result in erroneous reception. The maximum timing error which can be tolerated and yet still allow proper detection of an 11 bit ASCII character is then:

$$E_{\max} = \frac{0.5 \cdot \text{BIT TIME}}{\text{CHARACTER TIME}} - \frac{0.5P}{11P} = 4.5\%$$

where P is the period of single bit. The corresponding calculation for a 32 bit character yields:

$$E_{\max} = \frac{0.5P}{32P} = 1.6\%$$

Since this calculation does not allow for distortion on the signals, it is obvious that either extremely stable clocks will be required or a more tolerant algorithm must be devised. This problem is particularly serious at relatively high baud rates where the resolution of the counter (80μsecs with a 6 MHz crystal) becomes a significant percentage of the period of the received signal. At the 110 baud rate of the Teletype the 80μsec resolution of the clock allows a maximum accuracy of 0.33%; at 2400 baud this figure is reduced to 3.8%.

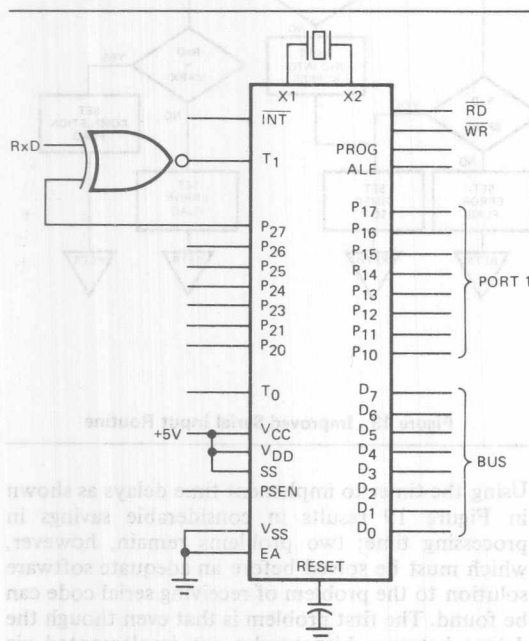


Figure 20. Detecting RxD Edges

Both efficient detection of the start bit and increased timing accuracy can be obtained if the MCS-48 can detect edges on the incoming received data (RxD). A hardware construct which allows this is shown in Figure 20.

The received data (RxD) is Exclusive NORed with bit seven of port two and fed into the TEST (T1) pin of the MCS-48. By manipulating P27 the program can now cause T1 to be either RxD or RxD. (If P27 = 1 then T1 = RxD; if P27 = 0 then T1 = RxD.) Note that not only can T1 be tested directly by the software but that it is the input which is used when the MCS-48 timer is in the event counter mode. The significance of this will be discussed later. The relationship between T1, P27, and RxD is given by the Boolean expression:

$$T1 = P27 \cdot \overline{RxD} + \overline{P27} \cdot RxD$$

Figure 21 flowcharts a means of utilizing this hardware construct to avoid the necessity of wasting time in program loops to detect the leading edge of the start bit. The receive operation is initialized when the program desiring to receive serial data calls the INIT subroutine (Figure 21a). Since INIT is going to manipulate the timer the first action it performs is to disable the timer overflow interrupt. Its next step is to set P27 to a logical 1. Setting P27 in this manner causes the TEST 1 input to the MCS-48 to follow RxD. By setting up the receive circuitry in this manner a high to low transition will occur on TEST 1 when the RxD goes from the MARKING to SPACING state (i.e. the START

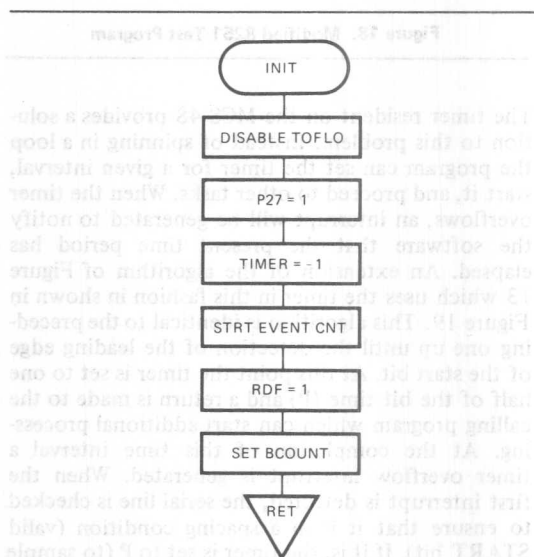


Figure 21a. Interrupt Driven Serial Receive Flowchart

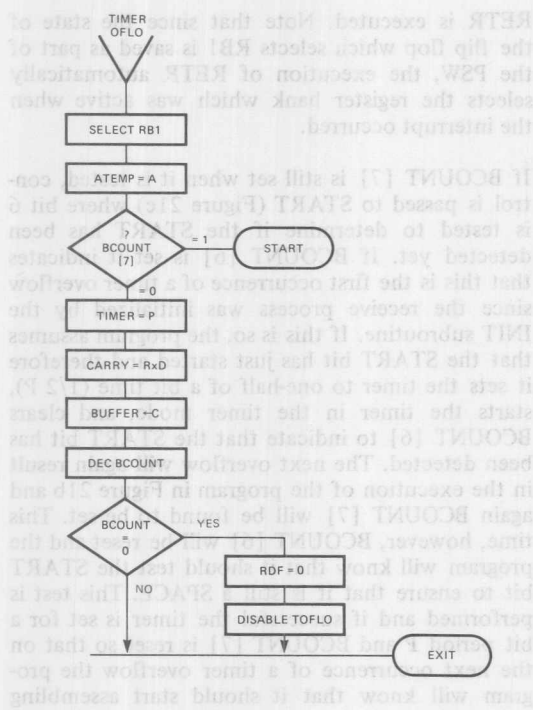


Figure 21b. Interrupt Driven Serial Receive Flowchart

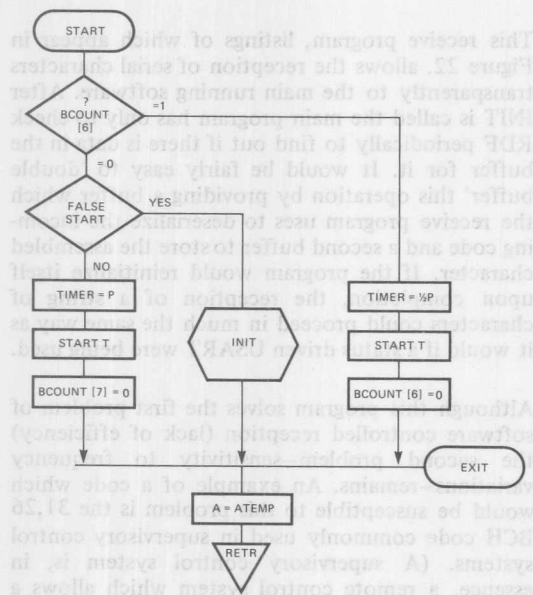
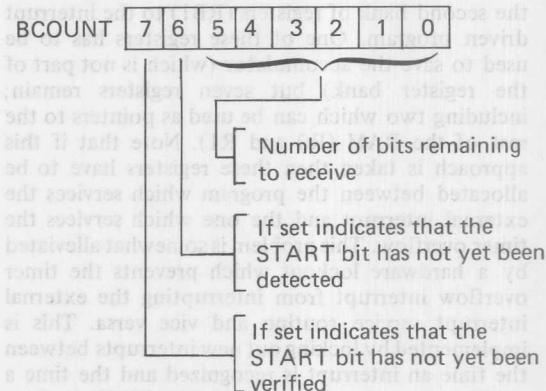


Figure 21c. Interrupt Driven Serial Receive Flowchart

bit occurs). By setting the timer to 0FFH and enabling it in the event count mode, the INIT routine sets up the MCS-48 to generate a timer overflow interrupt on the next MARK to SPACE transition of RxD (the TEST 1 input doubles as the event counter input). Before returning to the calling program the INIT routine sets a flag (RDF) which will be cleared by the receive program when the requested receive operation is complete. INIT also sets a value into a register called BCOUNT. The receive program interprets BCOUNT as follows:



In order to request the reception of the 11 bit ASCII code INIT would set BCOUNT to 11001011B. The start bit has been neither verified nor detected and 11 bits (1011B) are required.

After INIT is called the reception of the individual serial data bits will proceed on an interrupt driven basis until a complete character has been assembled. When this occurs the interrupt driven program will set the RDF (Receive Done Flag) to a zero to indicate that it has completed the requested operation and then terminate itself. The procedure which is used to accomplish this is shown in Figures 21b and 21c.

Since all operations of this program are the result of the occurrence of a timer overflow interrupt, it is necessary to briefly review the interrupt structure of the MCS-48. There are two sources of interrupt; an external interrupt which is the result of a logical zero signal applied to the INT pin of the MCS-48, and an internal interrupt which is caused by a timer overflow condition. The timer overflow occurs whenever the timer is incremented from 0FFH to zero whether it be in the timer or event count mode. When one of these events occurs the hardware in the MCS-48 forces the execution of a CALL. This CALL has a preset address of location 3 if it is due to the external interrupt and location 7 if it is due to a timer overflow. If both of these

the contents of the program counter for the running program and its PSW (program status word) on a stack the hardware maintains in RAM locations 8-23. Although the hardware saves the program counter and PSW, it remains the responsibility of any interrupt driven software to make absolutely certain that it does not modify any memory locations or registers which are being used by the main program. The most convenient way of ensuring this in the MCS-48 is to dedicate the second bank of registers (RB1) to the interrupt driven program. One of these registers has to be used to save the accumulator (which is not part of the register bank) but seven registers remain; including two which can be used as pointers to the rest of the RAM (R0 and R1). Note that if this approach is taken then these registers have to be allocated between the program which services the external interrupt and the one which services the timer overflow. This problem is somewhat alleviated by a hardware lockout which prevents the timer overflow interrupt from interrupting the external interrupt service routine and vice versa. This is implemented by locking out new interrupts between the time an interrupt is recognized and the time a RETR instruction is executed. The RETR instruction is like a normal RET (return from subroutine) except that the PSW as well as the program counter is restored. The RETR instruction can be very much thought of as a return from interrupt instruction in the MCS-48.

The receive program under discussion uses register bank 1 in the manner described. Whenever a timer overflow occurs (e.g. on the next MARK to SPACE transition of RxD after INIT is called), control is passed (by the hardware generated CALL) to the point labeled TIMER OFLO in Figure 21b. This program segment immediately selects register bank 1 (RB1) and then saves the accumulator (A) in a location called ATEMP which is actually R7 of RB1. The program then tests bit seven of BCOUNT (R6 of RB1) to find out if a START bit has been verified (i.e. the edge of the START bit has first been detected and then verified to still be a SPACE one-half a bit time later. If BCOUNT [7] is a zero the START has been verified and the program proceeds to set the timer to P (the period of the serial bit), get the current serial data into the carry bit, and then shift the carry bit into a buffer. After saving the data the program decrements BCOUNT and tests it for zero. If BCOUNT is zero the receive operation is complete so the program sets RDF to a zero and disables timer overflow interrupts. Whether or not BCOUNT is zero, control is passed to EXIT where A is loaded with ATEMP and a

the PSW, the execution of RETR automatically selects the register bank which was active when the interrupt occurred.

If BCOUNT [7] is still set when it is tested, control is passed to START (Figure 21c) where bit 6 is tested to determine if the START has been detected yet. If BCOUNT [6] is set it indicates that this is the first occurrence of a timer overflow since the receive process was initialized by the INIT subroutine. If this is so, the program assumes that the START bit has just started and therefore it sets the timer to one-half of a bit time ($1/2 P$), starts the timer in the timer mode, and clears BCOUNT [6] to indicate that the START bit has been detected. The next overflow will again result in the execution of the program in Figure 21b and again BCOUNT [7] will be found to be set. This time, however, BCOUNT [6] will be reset and the program will know that it should test the START bit to ensure that it is still a SPACE. This test is performed and if successful the timer is set for a bit period P and BCOUNT [7] is reset so that on the next occurrence of a timer overflow the program will know that it should start assembling serial bits into a character. If the test is unsuccessful, the subroutine INIT is used to reinitialize the receive program. In either case control is passed to EXIT where a return from interrupt mode occurs.

This receive program, listings of which appear in Figure 22, allows the reception of serial characters transparently to the main running software. After INIT is called the main program has only to check RDF periodically to find out if there is data in the buffer for it. It would be fairly easy to 'double buffer' this operation by providing a buffer which the receive program uses to deserialize the incoming code and a second buffer to store the assembled character. If the program would reinitialize itself upon completion, the reception of a string of characters could proceed in much the same way as it would if a status driven USART were being used.

Although this program solves the first problem of software controlled reception (lack of efficiency) the second problem—sensitivity to frequency variations—remains. An example of a code which would be susceptible to this problem is the 31,26 BCH code commonly used in supervisory control systems. (A supervisory control system is, in essence, a remote control system which allows a human or computer operator the control of a system via a serial communications link.) The BCH codes are used because of their error detection capabilities and are a class of cyclical redundancy

LOC	OBJ	SEQ	SOURCE STATEMENT	LOC	OBJ	SEQ	SOURCE STATEMENT
1		0	*****	0023	FE	71	START: MOV A,BCOUNT
2		1	*****	0024	D237	72	JB6 SLLC
3		2	*****			73	DO: IF TEST1=0 THEN
4		3	SERIAL INPUT USING THE MCS-48	0026	5635	75	JT1 SLLD
5		4	THIS CODE ASSUMES HARDWARE			76	DO: TIMER=P;
6		5	SHOWN IN FIG 20. TO USE			77	START TIMER;
7		6	THIS ROUTINE CALL INIT.			78	P27=0;
8		7	WHEN RDX=0 THE ASSEMBLED			79	EN I
9		8	CHARACTER WILL BE IN SERBUF			80	BCOUNT(7)=0;
10		9	*****			81	END;
11		10	*****	0028	23D7	83	MOV A,#-P
12		11	*****	002A	62	84	MOV T,A
13		12	*****	002B	55	85	STRT T
14		13	EQUATES	002C	9A7F	86	ANL P2,#7FH
15		14	*****	002E	05	87	EN I
16		15	*****	002F	FE	88	MOV A,BCOUNT
0007		16	ATMP EQU R7 ; STORAGE FOR A DURING INTERRUPT	0030	537F	89	ANL A,#7FH
0006		17	BCOUNT EQU R6 ; CONTAINS NUMBER OF BITS IN MSG	0032	AB	90	MOV BCOUNT,A
0002		18	COUNT EQU R2 ; UTILITY COUNTER	0033	043F	91	JMP SEXIT
0008		19	RXB EQU R0 ; POINTER			92	ELSE
0008		20	BITNO EQU 8 ; NUMBER OF BITS			93	DO:
0029		21	P EQU 41 ; SAMPLE PERIOD			94	CALL INIT;
0020		22	SERBUF EQU 204H ; SERIAL BUFFER			95	END;
0024		23	RDX EQU 24H ; RECEIVE DONE FLAG	0035	1441	96	SLLD: CALL INIT
		24	*****			97	ELSE
		25	*****			98	DO:
		26	CONTROL PASSED HERE WHEN TIMER OFLO OCCURS			99	TIMER=P/2;
		27	*****			100	START TIMER;
		28	*****			101	BCOUNT(6)=0;
0007		29	*****			102	END;
0007	D5	31	IMVEC: SEL RB1 ; /*ENTER INTERRUPT MODE*/	0037	23EC	103	SLLC: MOV A,#-(P/2)
0008	AF	32	MOV ATEMP,A	0039	62	104	MOV T,A
		33	IF BCOUNT(7)=0 THEN	003A	55	105	STRT T
0009	FE	34	MOV A,BCOUNT	003B	FE	106	MOV A,BCOUNT
000A	F223	35	JB7 START	003C	53BF	107	ANL A,#0BFH
		36	DO:	003E	AE	108	MOV BCOUNT,A
		37	TIMER=P;			109	END;
000C	23D7	38	MOV A,#-P	003F	FF	111	SEXT: MOV A,ATEMP
000E	G2	39	MOV T,A	0040	93	112	RETR
		40	START TIMER			113	*****
000F	55	41	SLLB: STRT T			114	*****
		42	/*CARRY-RXD*/			115	INITIALIZE ROUTINE-
		43	CARRY=P27 XOR TEST1;			116	STARTS RECEIVE PROCESS
0010	8A	44	IN A,P2			117	*****
0011	F7	45	RLC A			118	*****
0012	5615	46	JT1 TISR0			119	INIT:
0014	A7	47	CPL C			120	PROCEDURE:
		48	/*SHIFT CARRY INTO BUFFER*/			121	DO:
		49	RXB=SERBUF;			122	DISABLE INTERRUPTS;
		50	RSHT MEM(RXB);			123	P27=1;
0015	B020	51	TISR0: MOV RXB,#SERBUF			124	TIMER=1;
0017	28	52	SLOOP: XCH A,@RXB			125	START EVENT COUNT;
0018	67	53	RRC A			126	RDX=1;
0019	28	54	XCH A,@RXB			127	BCOUNT=BCBH OR BITNO
		55	IF BCOUNT=0 THEN			128	END;
001A	EE3F	56	DJNZ BCOUNT,SEXT			129	END INIT;
		57	DO:	0041	35	130	INIT: DIS TCNT1
		58	RDX=0;	0042	0A00	131	ORL P2,#00H
		59	DISABLE EX INT;	0044	23FF	132	MOV A,#-1
		60	END;	0046	62	133	MOV T,A
001C	B024	62	MOV RXB,#RDX	0047	45	134	STRT CNT
001E	27	63	CLR A	0048	B024	135	MOV @RXB,#RDX
001F	A8	64	MOV @RXB,A	004A	B001	136	MOV @RXB,#01H
0020	35	65	DIS TCNT1	004C	B01E	137	MOV @RXB,#1EH
		66	END;	004E	B0CB	138	MOV @RXB,#(BCBH OR BITNO)
0021	043F	67	JMP SEXIT	0050	25	139	EN TCNT1
		68	ELSE	0051	83	140	RET
		69	DO:			141	END OF PROGRAM
		70	IF BCOUNT(6)=0 THEN			142	*****
			*****			143	END

Figure 22. Interrupt Driven Serial Receive Program

codes such as those used in synchronous data communications (e.g. BISYNC or SDLC). BCH codes, named for their originators Bose, Chaudhuri, and Hocquenghem, are characterized by having a length of $n=2^m-1$. The number of redundant check bits can be mt where t is a positive integer (clearly $mt \leq n$). The 31,26 code fits this format with $m=5$ and $t=1$. The length of each message is $n=2^5-1=31$ with $5*1$ redundant bits, leaving 26 bits available for data transmission. With an appropriate poly-

nominal BCH codes can detect all errors consisting of $2t$ error bits and all burst errors of mt or fewer bits. The 31,26 BCH code will therefore detect any erroneous messages with 1 or 2 errors or bursts of errors of less than 5 bits. The 31,26 format (shown in Figure 23) requires the reception of a start bit followed by 31 information bits, clearly beyond the capability of the USART but perhaps within reach of a program controlled approach using the MCS-48 itself.

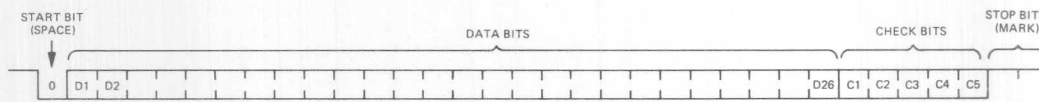


Figure 23. 31,26 BCH Code

A concept which reduces sensitivity to frequency deviations and thus allows the reception of longer codes is shown pictorially in Figure 24. The first line of this timing chart shows an alternative ones and zeros pattern on the RxD with a period of 5 milliseconds. The second line shows that by sampling at a period of exactly 5 milliseconds the data can be properly interpreted. The third and fourth lines show the effects of sampling with a period of six and four milliseconds respectively. In either case, an error occurs at the third sample where both periods result in sampling on an edge of the RxD signal. The third line of Figure 24 shows a hybrid sampling scheme which, based on some additional information, switches sampling periods between the two values. As can be seen in Figure 24, the data is sampled with a 4 millisecond period until the sampling begins to fall behind the data; at this point the sampling period is increased to six milliseconds and the sampling first catches up and then passes the center point of the data. As soon as this happens, the sampling period reverts to the 4 millisecond period and the cycle repeats. It can be seen that this scheme sets up a pattern which repeats indefinitely and the data can be successfully sampled. Note that the sampling pattern established is alternating periods of four and six milliseconds. The average period of this pattern, as might be expected, is 5 msec. Line 5 of Figure 24 shows the effect of a change in transmission speed to a period of 5.5 msec with no change in the sampling time. The sampling is again successful but the new sampling pattern is 4-6-6-6; 4-6-6-6, etc. Note that the average sample is again equal to the period of the received data (5.5). While this scheme

does seem to work, the question of what additional information is needed remains.

The MCS-48 must somehow decide when it is drifting out of synchronization and take corrective action. By referring back to Figure 24 it can be seen that if the MCS-48 could determine where the edges of RxD occurred with respect to its sampling times then the additional information would be available. As can be seen in the figure the choice of sampling period can be based on the following rule:

If an edge on the RxD line occurs during the first half of the current sampling period, then use the short period for the next sample. If an edge occurs during the second half of the period, then use the long sampling period for the next sample.

If the data on the RxD line does not change, of course, the MCS-48 will drift out of synchronization just as the original algorithm did. As long as edges occur on TxD, however, synchronization can be maintained. To maximize the allowable time between edges, the following addition could be made to the above rule:

If no edge occurs on the RxD line during a sample, then change sampling period from short to long or vice versa.

Note that this addition to the rule will result in using an average of the two sampling periods when no edge occurs for several bit times.

The edges of RxD can be easily detected by the use of the same structure (the Exclusive – NOR gate) which was added to the MCS-48 in Figure 20. This gate, which is used to detect the edge on RxD which begins the START bit, can naturally be used to detect any edge. Since the timer is being used to time the bit period, however, the event count input (T1) is not useful during the receive itself. By connecting the output of this gate, however, to the INT input to the MCS-48 (see Figure 25) it is possible to detect edges on RxD with the event counter when the program is trying to detect the START bit and by the external interrupt when the program is using the timer to control the sampling times.

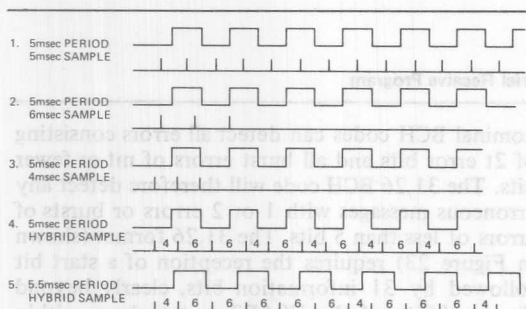


Figure 24. Various Sampling Alternatives

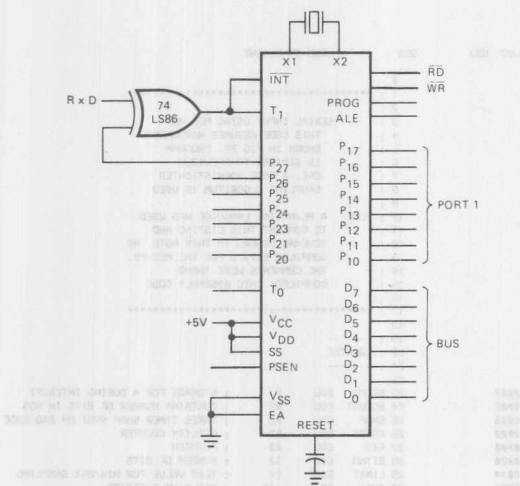
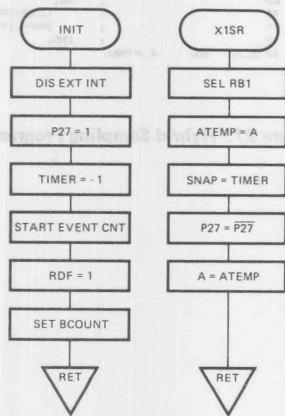


Figure 25. Modified Edge Detection

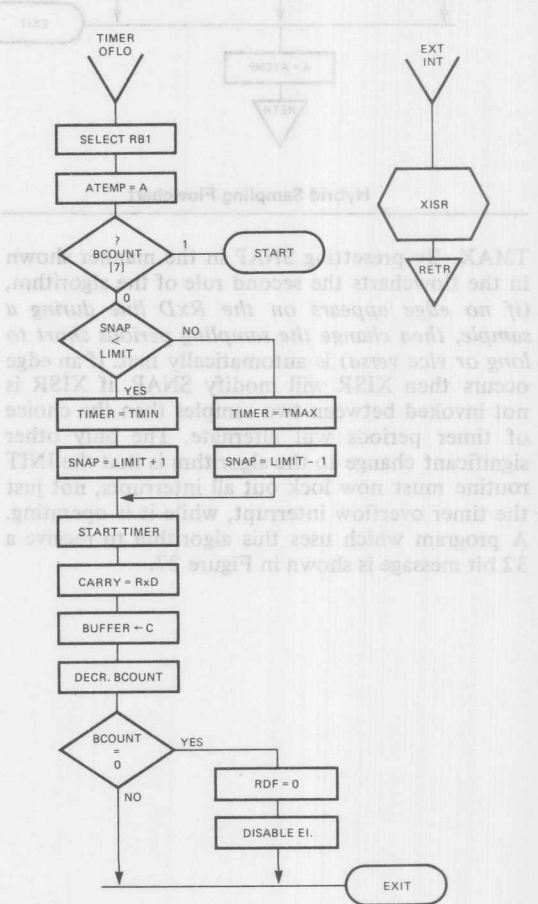
A modification to the program of Figure 21 which implements this new sampling algorithm is shown in Figure 26. The first deviation from the original program is the addition of a routine (XISR, Figure 26a which is called when an external interrupt occurs (i.e. when an edge occurs on RxD). This routine saves the status of the running program and then stores the current value of the timer register in a location called SNAP (R5 of RB1). After doing these operations the program complements bit 7 of port 2. Manipulating P27 in this manner will cause the Exclusive NOR gate to turn off the external interrupt and will set it up to generate another interrupt when the RxD line changes again (has another edge).



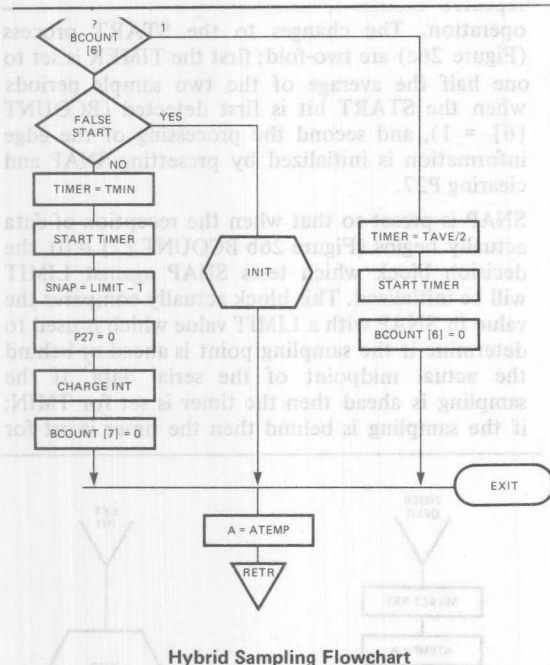
Hybrid Sampling Flowchart

Because of this edge detection it is important to condition RxD with hardware filters to ensure that the edges of RxD are clean. Any ringing will cause repeated CALLs to XISR and probable erroneous operation. The changes to the START process (Figure 26c) are two-fold; first the TIMER is set to one half the average of the two sample periods when the START bit is first detected (BCOUNT [6] = 1), and second the processing of the edge information is initialized by presetting SNAP and clearing P27.

SNAP is preset so that when the reception of data actually begins (Figure 26b BCOUNT [7] = 0), the decision block which tests SNAP against LIMIT will be initialized. This block actually compares the value in SNAP with a LIMIT value which is used to determine if the sampling point is ahead or behind the actual midpoint of the serial data. If the sampling is ahead then the timer is set for TMIN; if the sampling is behind then the timer is set for



Hybrid Sampling Flowchart



TMAX. By presetting SNAP in the manner shown in the flowcharts the second rule of the algorithm, (if no edge appears on the Rx/D line during a sample, then change the sampling periods short to long or vice versa) is automatically met. If an edge occurs then XISR will modify SNAP, if XISR is not invoked between two samples then the choice of timer periods will alternate. The only other significant change to the algorithm is that the INIT routine must now lock out all interrupts, not just the timer overflow interrupt, while it is operating. A program which uses this algorithm to receive a 32 bit message is shown in Figure 27.

```

1
2
3 SERIAL INPUT USING MCS-48
4 THIS CODE ASSUMES HARDWARE
5 SHOWN IN FIG 25. PROGRAM
6 IS SIMILAR TO PREVIOUS
7 ONE. A MORE SOPHISTICATED
8 SAMPLING ALGORITHM IS USED
9
10 NOTE: A PL/M LIKE LANGUAGE WAS USED
11 TO COMMENT THIS LISTING AND
12 SEVERAL OTHERS IN THIS NOTE. NO
13 COMPILER EXISTS FOR THE MCS-48.
14 THE COMMENTS WERE "HAND
15 COMPILED" INTO ASSEMBLY CODE
16
17
18
19
20 EQUATES
21
22
23 ATEMP EQU R7 ; STORAGE FOR A DURING INTERRUPT
24 BCOUNT EQU R6 ; CONTAINS NUMBER OF BITS IN MSG
25 SNAP EQU R5 ; TAKES TIMER SNAP SHOT ON RXD EDGE
26 COUNT EQU R2 ; UTILITY COUNTER
27 RXD EQU R8 ; POINTER
28 BITNO EQU 32 ; NUMBER OF BITS
29 LIMIT EQU 28 ; TEST VALUE FOR MIN/MAX SAMPLING
30 TMAX EQU -43 ; MAX SAMPLE PERIOD
31 TMIN EQU -39 ; MINIMUM SAMPLE PERIOD
32 HALF EQU -28 ; HALF NOMINAL PERIOD
33 SERBUF EQU 28H ; START OF SERIAL BUFFER
34 RDF EQU 24H ; RECEIVE DONE FLAG
35
36
37 CONTROL PASSED HERE ON EXT. INT.
38
39
40 ORG 03H ; CALL SERVICE ROUTINE
41
42 EVEC: CALL XISR
43 RETR
44
45
46 CONTROL PASSED HERE WHEN TIMER OFLO OCCURS
47
48
49
50 THVEC: SEL RB1 ; /ENTER INTERRUPT MODE/
51 MOV ATEMP,A ; IF BCOUNT[7]=0 THEN
52 ; IF BCOUNT[7]=0 THEN
53 MOV A,BCOUNT
54 JB7 START
55 ; DO:
56 ; IF SNAP<LIMIT THEN
57 MOV A,SNAP
58 ADD A,#LIMIT
59 JB7 SLLA
60 ; DO:
61 ; TIMER=TMIN;
62 ; SNAP=LIMIT+1;
63 ; END;
64 MOV A,#TMIN
65 MOV T,A
66 MOV SNAP,#LIMIT-1
67 JPP SLLB
68 ; ELSE
69 ; DO:
70 ; TIMER=TMAX;
71 ; SNAP=LIMIT-1;
72 ; END;
73 SLLA: MOV A,#TMAX
  
```

Figure 27. Hybrid Sampling Program



```

LOC  OBJ  SEQ  SOURCE STATEMENT
0019 62 74 MOV T,A
001A BD13 75 MOV SNAP,#LIMIT-1
001C 55 76 SLLB: STRT T ; START TIMER;
001D 55 77 SLLB: STRT T ; CARRY+P27 XOR TEST1;
001E 7A 78 ; /*CARRY+RXD*/
001F 7A 79 ; CARRY+P27 XOR TEST1;
0020 7A 80 IN A,P2
0021 7A 81 RLC A,P2
0022 7A 82 TISRD A,P2
0023 7A 83 CPL C
0024 7A 84 ; /*SHIFT CARRY INTO BUFFER*/
0025 7A 85 ; RXB=SERBUF;
0026 7A 86 ; COUNT-4;
0027 7A 87 ; DO WHILE COUNT<0;
0028 7A 88 ; RSHFT MEM(RXD);
0029 7A 89 ; RXB=RXB-1;
002A 7A 90 ; COUNT=COUNT-1;
002B 7A 91 ; END;
002C B824 92 TISRD: MOV RXB,#SERBUF
002D B824 93 MOV COUNT,#4
002E 28 94 SLOOP XCH A,RXB
002F 67 95 RRC A
0030 28 96 XCH A,@RXB
0031 18 97 INC RXB
0032 EA26 98 DJNZ COUNT,SLOOP
0033 99 ; BCOUNT=BCOUNT-1;
0034 99 ; IF BCOUNT=0 THEN
0035 EE54 100 DJNZ BCOUNT,SEXIT
0036 101 ; DO;
0037 102 ; RDF=#;
0038 103 ; DISABLE EX INT;
0039 104 ; END;
003A B824 105 MOV RXB,#RDF
003B 27 106 CLR A
003C A8 107 @RXB,A
003D 35 108 DIS TCNT1
003E 15 109 DIS I
003F 8454 110 JMP EXIT ; END;
0040 111 ; ELSE
0041 112 ; DO;
0042 113 ; IF BCOUNT(6)=0 THEN
0043 FE 114 START: MOV A,BCOUNT
0044 D24C 115 JBG SLLC ; DO;
0045 116 ; IF TEST1=0 THEN
0046 564A 117 JT1 SLLD ; DO;
0047 118 ; TIMER=TMIN;
0048 119 ; START TIMER;
0049 120 ; SNAP=LIMIT+1;
0050 121 ; P27=0;
0051 122 ; EN I
0052 123 ; BCOUNT(7)=0;
0053 124 ; END;
0054 23D9 125 MOV A,#TMIN
0055 62 126 MOV T,A
0056 55 127 STRT T
0057 BD15 128 MOV SNAP,#LIMIT+1
0058 9A7F 129 ANL P2,#7FH
0059 85 130 EN I
005A FE 131 MOV A,BCOUNT
005B 537F 132 ANL A,#7FH
005C AE 133 MOV BCOUNT,A
005D 8454 134 JMP EXIT
005E 135 ; ELSE
005F 136 ; DO;
0060 137 ; CALL INIT;
0061 138 ; END;

```

```

LOC  OBJ  SEQ  SOURCE STATEMENT
0062 1456 143 SLLD: CALL INIT
0063 144 ; ELSE
0064 145 ; DO;
0065 146 ; TIMER=(TMIN+TMAX)/2;
0066 147 ; START TIMER;
0067 148 ; BCOUNT(6)=0;
0068 149 ; END;
0069 23EC 150 SLLC: MOV A,#HALF
006A 62 151 MOV T,A
006B 55 152 STRT T
006C FE 153 MOV A,BCOUNT
006D 53BF 154 ANL A,#0BFH
006E AE 155 MOV BCOUNT,A
006F 156 ; END;
0070 FF 157 SEXIT: MOV A,ATEMP
0071 93 158 RETR
0072 159 ; /*EXIT INTERRUPT MODE*/
0073 160 ;
0074 161 ; -----
0075 162 ; INITIALIZE ROUTINE-
0076 163 ; STARTS RECEIVE PROCESS
0077 164 ; -----
0078 165 ;
0079 166 ; INIT:
0080 167 ; PROCEDURE;
0081 168 ; DO;
0082 169 ; DISABLE INTERRUPTS;
0083 170 ; P27=1;
0084 171 ; TIMER=1;
0085 172 ; START EVENT COUNT;
0086 173 ; RDF=1;
0087 174 ; BCOUNT=BC0H OR BITNO
0088 175 ; END;
0089 176 ; END INIT;
0090 15 177 INIT: DIS I
0091 35 178 DIS TCNT1
0092 9A86 179 ORL P2,#0BH
0093 23FF 180 MOV A,#-1
0094 62 181 MOV T,A
0095 45 182 STRT CNT
0096 B824 183 MOV RXB,#RDF
0097 F9 184 MOV A,#1
0098 A8 185 MOV @RXB,A
0099 25 186 EN TCNT1
009A BEE0 187 MOV BCOUNT,#BC0H OR BITNO
009B 83 188 RET
009C 189 ;
009D 190 ; -----
009E 191 ; INTERRUPT SERVICE ROUTINE
009F 192 ; -----
009A 193 ;
009B 194 ; XISR:
009C 195 ; PROCEDURE;
009D 196 ; DO;
009E 197 ; /*ENTER INTERRUPT MODE*/
009F 198 ; SNAP=TIMER;
009A 199 ; P27=NOT P27;
009B 200 ; END XISR;
009C 201 ;
009D 202 ; XISR: SEL RB1
009E 203 MOV ATEMP,A
009F 204 MOV A,T
009A 205 MOV SNAP,A
009B 206 IN A,P2
009C 207 XRL A,#0BH
009D 208 OUTL P2,A
009E 209 MOV A,ATEMP
009F 210 RET
009A 211 ; END OF PROGRAM

```

Figure 27. Hybrid Sampling Program

TRANSMITTING SERIAL CODE

Serial transmission is conceptually far simpler than serial reception since no synchronization is required. All that is required is to use the timer to generate interrupts at the bit rate and present the character to be transmitted serially at an I/O pin. A program which does this is shown in Figure 28. The transmission of serial data becomes much more complicated if it must occur simultaneously with reception.

If both reception and transmission are to occur simultaneously then obviously contention will exist for the use of the timer. It is possible to allow the simultaneous reception and transmission of serial data using the timer as a general clock which controls software maintained timers. The attainable baud rates using such techniques are, however, limited and the use of a 8251 USART is probably

indicated in all but the most cost sensitive applications. An exception to this rule occurs when the system, although full duplex in nature, actually transmits the same data as it receives. An example of this is a microprocessor driving a terminal such as a Teletype. Although the circuit to the terminal is full duplex, the data that is transmitted is generally the same as that received. A minor modification to the program shown in Figure 26 would implement this mode of operation. The modification would be to the XISR routine and it would add the code necessary to place the TxD I/O pin in the same state as the RxD line. Since any change in RxD results in a call to XISR, this modification would cause the retransmission of any received data. Whenever it becomes necessary to transmit data which is not being received, the program of Figure 28 could be used in a half duplex manner.

LOC	OBJ	SEQ	SOURCE STATEMENT	LOC	OBJ	SEQ	SOURCE STATEMENT
0		0	START	000F 0A		37	IN A, P2
1		1	SETUP	0010 D380		38	XRL A, #80H
2		2	SERIAL TRANSMIT ON THE MCS48	0012 3A		39	OUTL P2, A
3		3	TO USE PUT A CHAR IN BUFF AND	0013 F619		40	JC BIT0N
4		4	SET CHARAV TO 8FH, WHEN THE	0015 5AEF		41	ANL P2, #CBIT
5		5	TRANSMITTER IS READY FOR ANOTHER	0017 841B		42	JMP EXIT
6		6	CHAR IT WILL CLEAR CHARAV, THE	0019 8A18		43	BIT0N: ORL P2, #SBIT
7		7	TRANSMISSION IS DOUBLE BUFFERED.	001B FF		44	EXIT: MOV A, ATEMP
8		8		001C 93		45	RETR
9		9				46	
10		10				47	
11		11	EQUATES			48	BIT ROUTINE
12		12				49	-PICKS THE NEXT BIT TO TRANSMIT
13		13				50	
14	ATEMP	14	EQU R7 ; STORAGE FOR A DURING INT.	001D FB		52	BIT: MOV A, COUNT
15	PTOS	15	EQU R6 ; PARALLEL TO SERIAL CONVERTER	001E C627		53	JZ IDLE
16	BUFF	16	EQU R5 ; CHARACTER BUFFER	0020 FE		54	MOV A, PTOS
17	CHARAV	17	EQU R4 ; CHARACTER AVAILABLE FLAG	0021 67		55	RRC A
18	COUNT	18	EQU R3 ; BIT COUNTER	0022 4388		56	ORL A, #80H
19	CBIT	19	EQU 8FH ; MASK TO CLEAR TXD IN P24	0024 AE		57	MOV PTOS, A
20	SBIT	20	EQU 010H ; MASK TO SET TXD IN P24	0025 CB		58	DEC COUNT
21	P	21	EQU 41H ; PERIOD OF TXD	0026 83		59	RET
22		22				60	
23		23				61	IDLE: CLR C
24		24	CONTROL PASSED HERE ON TIMER OVERFLOW	002B FC		62	MOV A, CHARAV
25		25				0029 962D	JNZ GOTONE
26	ORG	26	07H ; ENTER INTERRUPT MODE	002B A7		64	CPL C
27	TOFLO: SEL	27	RB1	002C 83		65	RET
28	MOV	28	ATEMP, A			66	
29		29				67	GOTONE: MOV A, BUFF
30		30				002E AC	MOV PTOS, A
31	MOV	31	A, #P ; SET TIMER FOR P	002F BB8A		69	MOV COUNT, #18
32	MOV	32	T, A	0031 BC88		70	MOV CHARAV, #8
33	STRT	33	T	0033 83		71	RET
34		34				72	
35	CALL	35	BIT ; GET BIT INTO CARRY			73	END ; END OF PROGRAM
36		36					

Figure 28. Serial Transmission

GENERATING PARITY

Many communications schemes require the generation and checking of parity. If a USART is used it can be programmed to automatically generate and check parity. If the communications is handled by software within the MCS-48™ then the program must perform parity calculations. Calculating parity is easy if one remembers what parity really means. A character has even parity if the number of one bits in it is even. A character has odd parity if it has an odd number of ones. The program segment shown in Figure 29 can be caused to calculate parity. It starts by setting a loop count to eight and

CONCLUSION

This Application Note has presented a very small sampling of the application techniques possible with the MCS-48™ family. The application of this new single chip computer system to tasks which have not yet yielded to the power of the micro-processor will present a fascinating challenge to the system designer.

LOC	OBJ	SEQ	SOURCE STATEMENT
		8	
		1	
		2	*****
		3	
		4	PARITY
		5	THIS PROGRAM GENERATES PARITY
		6	ON THE ACCUMULATOR
		7	CARRY WILL BE SET IF A HAS ODD PARITY
		8	
		9	*****
		10	
		11	
		12	-----
		13	EQUATES
		14	-----
		15	
0002		16	COUNT EQU R2
		17	
0100		18	PAR: ORG 100H
0100 BABB		19	MOV COUNT, #8 ; SET LOOP COUNT
0102 97		20	CLR C ; INITIALIZE CARRY
		21	
		22	
0103 77		23	LOOP: RR A ; FOR EACH ZERO BIT IN A
0104 1207		24	JB0 OVER ; COMPLEMENT THE CARRY FLAG
0106 A7		25	CPL C
		26	
		27	
		28	END ; END OF PROGRAM

Figure 29. Parity Generation

clearing the CARRY flag. After this initialization a loop is executed eight times. During each execution the accumulator is rotated and the least significant bit is tested. If the bit is a zero the CARRY flag is complemented, if the bit is a one no further action is taken. Since an even number of zeros implies an even number of ones for an eight bit character, after all eight loops have been accomplished the CARRY bit will be set if an odd number of ones were encountered; it will be reset if the number were even. Since the RR instruction does not involve CARRY the net result of executing this program loop is to set CARRY if parity is odd without effecting the character in the accumulator.

3-2	Introduction
3-3	Hardware Schematic
3-7	Software Listing
3-11	Configuration Examples
3-15	Utility Subroutines
3-25	Assembly Cross Reference

June 1978

Keyboard / Display Scanning With Intel®'s MCS-48™ Microcomputers

John Wharton
Microcomputer Applications

**Keyboard/Display Scanning
With Intel®'s MCS-48™
Microcomputers**

Contents

Introduction	3-3
Hardware Schematic	3-6
Software Listing	3-7
Configuration Equates	3-11
Utility Subroutines	3-18
Assembly Cross Reference	3-25

INTRODUCTION

This application note presents a software package for interfacing members of Intel's MCS-48™ family of single-chip microcomputers with keyboards and displays using a minimum of external components. Because of the similarity of the architectures of the various members of the family (the 8035, 8048, 8748, 8039, 8049, 8021, and 8022 microcomputers; also the 8041 and 8741 universal peripheral interfaces in the UPI-41® family), the code included here could run with minor modifications on any member of the family.

Since keyboard and display logic can be just one of several functions handled by a microprocessor, the added cost of including these functions in a system is minimal. In fact, considering the extremely low cost of standard X-Y matrix keyboards and integrated displays, their use is often more cost effective than even a handful of discrete switches and indicators. Thus, the additional flexibility of keyboard input and display output can be added to inexpensive consumer products (such as games, clocks, thermostats, tape recorders, etc.), while producing a net savings in system cost.

Since each potential application will have its own unique combination of keys and display characters, the program is written so that very little modification is needed to interface it with a wide variety of hardware configurations. In general, the only changes required are within the set of initial EQUates at the beginning of the program.

Along with the basic software for driving a multiplexed display and/or scanning and debouncing an X-Y matrix of key switches, a collection of utility subroutines is also included for implementing the most commonly used keyboard and display utility functions, such as copying simple messages onto the display or determining the encoded value of each key in the key matrix. As a result of the versatile architecture and applications-oriented instruction set of the MCS-48 family, the entire package fits into about 250 bytes of internal program ROM or EPROM, leaving the rest of the ROM space for the program to cook the perfect piece of toast, or whatever. By tailoring the software to match a known hardware configuration, or by selecting only those functions needed for a given application, the program size could be even further reduced.

Since what is being presented in this application note is a software package, rather than the usual hardware/software system design, the format of this note is somewhat different from most — it consists primarily of a long program listing reproduced in the following pages. For the most part, the listing is self-explanatory, with comments introducing each subroutine and major code segment. Some parts of this introduction are reproduced in the program listing itself, explaining the configuration of the prototype system. However, an additional bit of explanation would make the listing easier to understand, especially for those readers unfamiliar with the concept of multiplexed displays and keyboards.

In traditional digital system design, various hardware registers or counters were used to hold binary or BCD values which had to be conveyed to the user. The standard way of presenting this information was by connecting each register to a seven-segment encoder (such as the 7447) driving a single display character, as represented by Figure 1. Thus, two ICs, seven current limiting resistors, and about 45 solder joints were required for each digit of output. Consider how traditional techniques might be (mis-)applied in designing a microprocessor system: the designer could add a latch, encoder, and resistors for each digit of the display. Still another latch and decoder could be used to turn on one of the decimal points (if used). The characters displayed could only be a sequence of decimal digits. In the same vein, a large matrix of key switches could be read by installing an MSI TTL priority encoder read by an additional input port. Not only would all this use a lot of extra I/O ports and increase the system price and part count drastically, but the flexibility and reliability of the system would be greatly reduced.

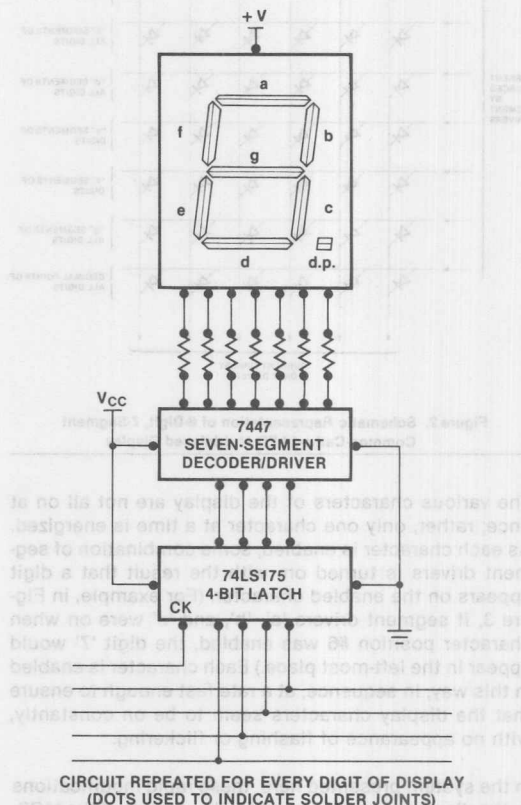


Figure 1. Wrong Way to Design Multiple Digit Displays for Microcomputer Systems

Instead, a scheme of time-multiplexing the display can be used to decrease costs, part count, and interconnections, while allowing a wider range of character types to be used on the display. The techniques used here are fairly typical of today's integrated subsystems designed especially for controlling keyboards and displays (such as in calculators or the Intel® 4269, 8278, and 8279 Keyboard/Display Controller Devices).

In a multiplexed display, all the segments of all the characters are interconnected in a regular two-dimensional array. One terminal of each segment is in common with the other segments of the same character; the other terminal is connected with the same segments of the other characters. This is represented schematically in Figure 2. A digit driver or segment driver is needed for each of these common lines.

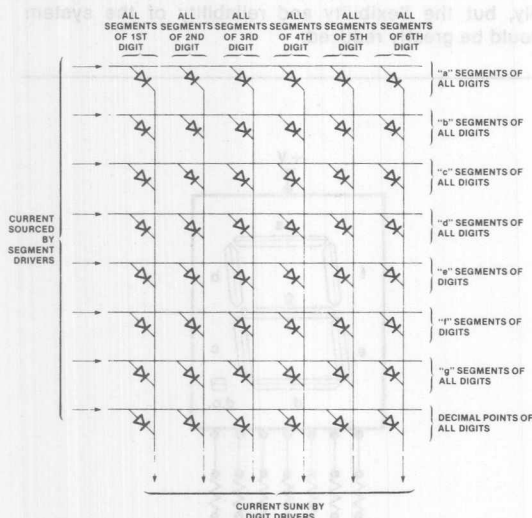


Figure 2. Schematic Representation of 6-Digit, 7-Segment Common-Cathod LED Multiplexed Display

The various characters of the display are not all on at once; rather, only one character at a time is energized. As each character is enabled, some combination of segment drivers is turned on, with the result that a digit appears on the enabled character. (For example, in Figure 3, if segment drivers 'a', 'b', and 'c' were on when character position #6 was enabled, the digit '7' would appear in the left-most place.) Each character is enabled in this way, in sequence, at a rate fast enough to ensure that the display characters seem to be on constantly, with no appearance of flashing or flickering.

In the system presented here, these rapid modifications to the display are all made under the control of the MCS-48™ microcomputer. At periodic intervals the computer quickly turns off all display segments, disables the character now being displayed and enables the next, looks up the pattern of segments for the next character

to be displayed, and turns on the appropriate segments. With the next character now turned on, the processor may now resume whatever it had been doing before. The whole display updating task consumes only a small fraction of the processor's time.

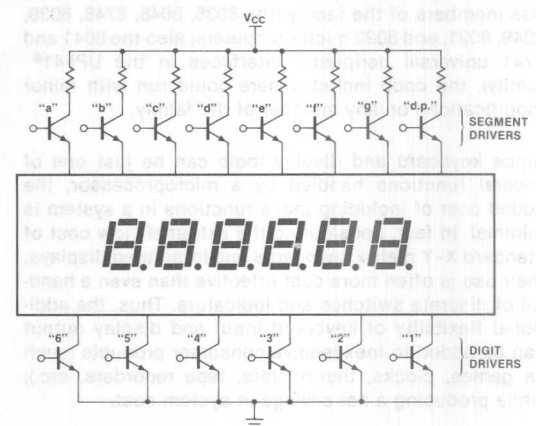


Figure 3. Segment and Digit Drivers used with 6-Position, 7-Segment LED Display

Moreover, since the computer rather than a standard decoder circuit is used to turn the segments off and on, patterns for characters other than decimal digits may be included in the display. Hexadecimal characters, special symbols, and many letters of the alphabet are possible. With sufficient imagination this feature can be exploited for some applications, as suggested by the examples in Figure 4.

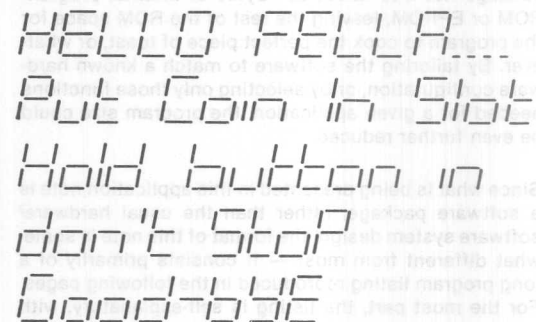


Figure 4. Examples of Typical Messages Possible with Simple 7-Segment Displays

As each character of the display is turned on, the same signal may be used to enable one row of the key matrix. Any keys in that row which are being pressed at the time will then pass the signal on to one of several "return lines", one corresponding to each column of the matrix. (See Figure 5.) By reading the state of these control lines, and knowing which row is enabled, it is possible to compute which (if any) of the keys are down. Note that the keys need not be physically arranged in a rectangular array; Figure 5 is merely a schematic.

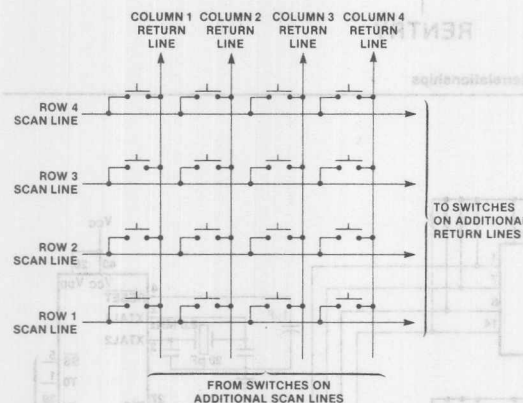


Figure 5. Schematic of X-Y Matrix Multiplexed Keyboard

Since each character is on for only a small fraction of the total display cycle, its segments must be driven with a proportionately higher current so that their brightness averages out over time. This requires character and segment drivers which can handle higher than normal levels of current. Various types of drivers can be used, ranging from specially designed circuits to integrated or discrete transistor arrays. The selection depends on several factors, including the type of display being used (LED, vacuum fluorescent, neon, etc.), its size, the number of characters, and the polarity of the individual segments. Some drivers have active high inputs, some active low. Some invert their input logic levels, some do not. Some require insignificant input currents, some present a considerable load. Some systems use external logic to enable one of N characters or to produce the appropriate segment pattern for a given digit, some systems implement these functions through software.

Because of these and the other variables which make each application unique, provisions are made in the first page of symbol EQUates to allow the user to specify such things as the number of characters in the display or the polarity of the drivers used, and the program will be assembled accordingly. The display is refreshed on each timer interrupt, which occurs every $32 \times (\text{TICK})$

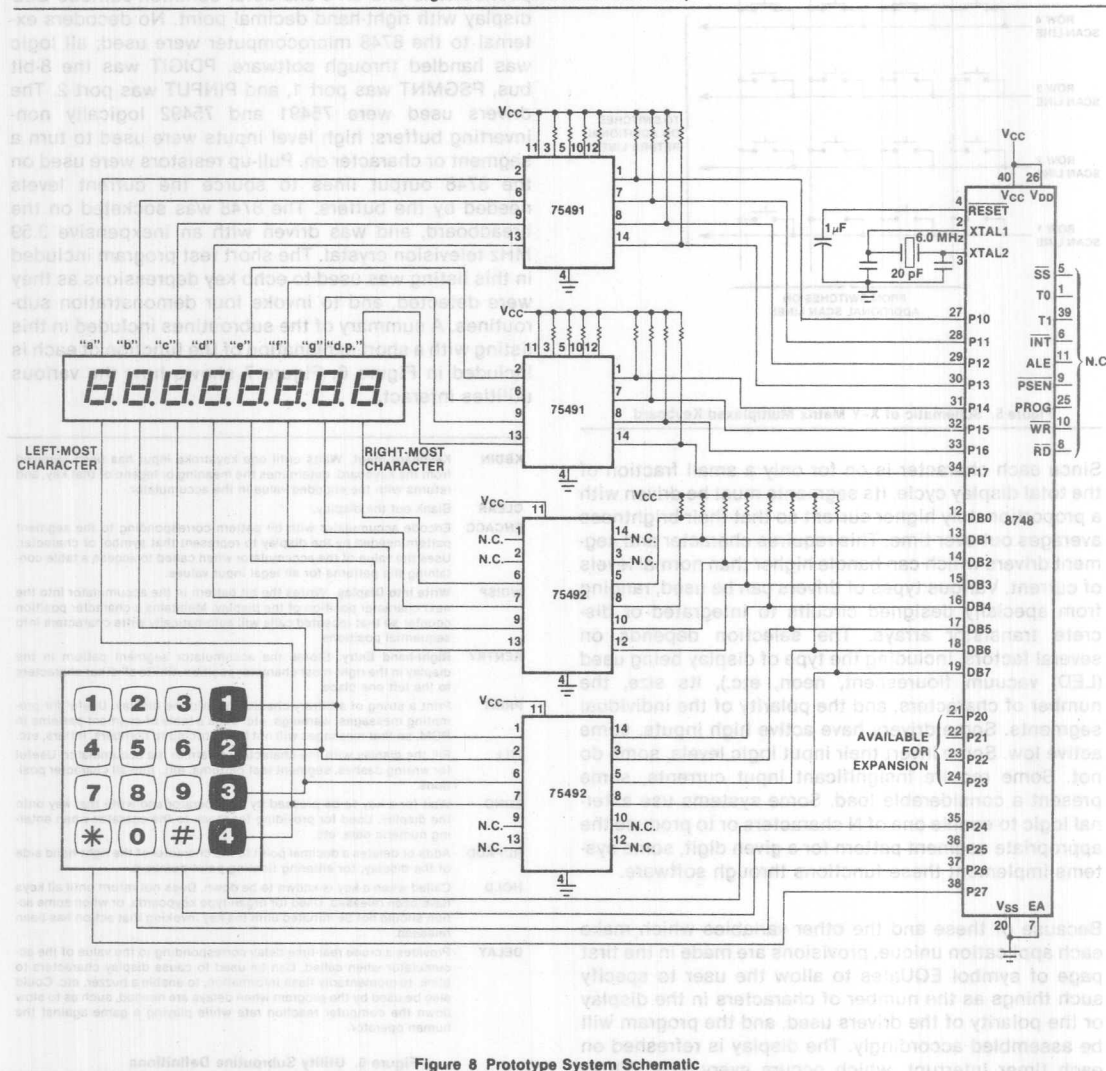
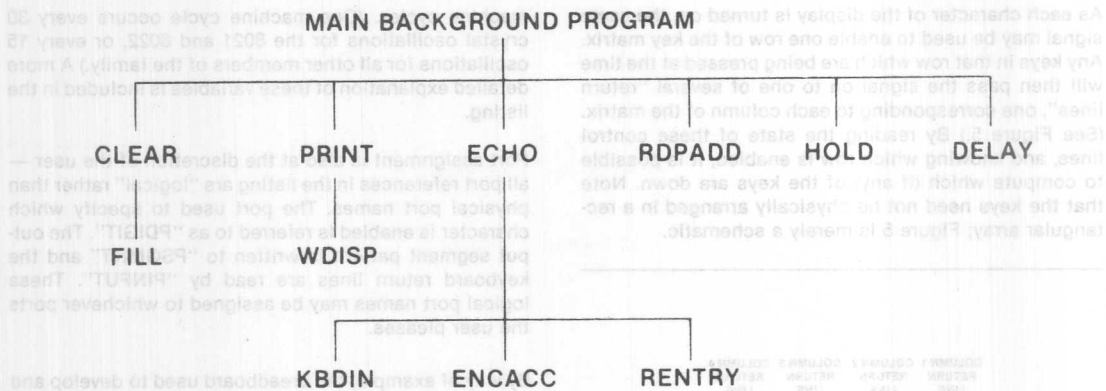
machine cycles. (One machine cycle occurs every 30 crystal oscillations for the 8021 and 8022, or every 15 oscillations for all other members of the family.) A more detailed explanation of these variables is included in the listing.

Port assignment is also at the discretion of the user — all port references in the listing are "logical" rather than physical port names. The port used to specify which character is enabled is referred to as "PDIGIT". The output segment pattern is written to "PSGMNT" and the keyboard return lines are read by "PINPUT". These logical port names may be assigned to whichever ports the user pleases.

By way of example, the breadboard used to develop and debug this software used a matrix of 16 single-pole pushbuttons and an 8-character common-cathode LED display with right-hand decimal point. No decoders external to the 8748 microcomputer were used; all logic was handled through software. PDIGIT was the 8-bit bus, PSGMNT was port 1, and PINPUT was port 2. The drivers used were 75491 and 75492 logically non-inverting buffers: high level inputs were used to turn a segment or character on. Pull-up resistors were used on the 8748 output lines to source the current levels needed by the buffers. The 8748 was socketed on the breadboard, and was driven with an inexpensive 3.59 MHz television crystal. The short test program included in this listing was used to echo key depressions as they were detected, and to invoke four demonstration subroutines. A summary of the subroutines included in this listing with a short explanation of the function of each is included in Figure 6; Figure 7 shows how the various utilities interact.

KBDIN	Keyboard Input. Waits until one keystroke input has been received from the keyboard; determines the meaning or legend of that key, and returns with the encoded value in the accumulator.
CLEAR	Blank out the display.
ENCACC	Encode accumulator with bit pattern corresponding to the segment pattern needed by the display to represent that symbol or character. Uses the value of the accumulator when called to access a table containing the patterns for all legal input values.
WDISP	Write into Display. Writes the bit pattern in the accumulator into the next character position of the display. Maintains a character position counter so that repeated calls will automatically write characters into sequential positions.
REENTRY	Right-hand Entry. Stores the accumulator segment pattern in the display in the right-most character position. Shifts all other characters to the left one place.
PRINT	Print a string of arbitrary characters onto the display. Useful for prompting messages, warnings, etc. Uses a table of segment patterns in ROM, so that messages will not be restricted to numbers, letters, etc.
FILL	Fill the display with the character pattern in the accumulator. Useful for writing dashes, segment test patterns, etc., into all character positions.
ECHO	Wait for a key to be pressed by the operator and write that key onto the display. Used for providing feedback to the operator when entering numeric data, etc.
RDPADD	Adds or deletes a decimal point to the character at the right-hand side of the display, for entering floating point numbers.
HOLD	Called when a key is known to be down. Does not return until all keys have been released. Used for organ-type keyboards, or when some action should not be initiated until the key invoking that action has been released.
DELAY	Provides a crude real-time delay corresponding to the value of the accumulator when called. Can be used to cause display characters to blink, to momentarily flash information, to enable a buzzer, etc. Could also be used by the program when delays are needed, such as to slow down the computer reaction rate while playing a game against the human operator.

Figure 6. Utility Subroutine Definitions



ISIS-II MCS-48/UP1-41 MACRO ASSEMBLER, V2.0
AP40: INTEL MCS-48 KEYBOARD/DISPLAY APPLICATION NOTE APPENDIX

LOC	OBJ	SEQ	SOURCE STATEMENT
		1	\$MACROFILE XREF
		2	\$TITLE('AP40: INTEL MCS-48 KEYBOARD/DISPLAY APPLICATION NOTE APPENDIX')
		3	
		4	THE FOLLOWING SOFTWARE PACKAGE PROVIDES A SEVEN SEGMENT DISPLAY
		5	INTERFACE FOR MICROCOMPUTERS IN THE INTEL MCS-48 FAMILY.
		6	THE CODE IS WRITTEN SO THAT VARIOUS HARDWARE
		7	CONFIGURATIONS CAN BE ACCOMMODATED BY REDEFINING THE INITIAL VARIABLES.
		8	IN MOST SITUATIONS, THE KEYBOARD/DISPLAY INTERFACE WILL BE REQUIRED TO
		9	IMPLEMENT MORE SOPHISTICATED SINGLE-CHIP SYSTEMS (CALCULATORS, SCALES, CLOCKS,
		10	ETC.), WITH SECTIONS OF THE FOLLOWING CODE SELECTED AND MODIFIED AS NECESSARY
		11	FOR EACH APPLICATION.
		12	
		13	A SINGLE SUBROUTINE (CALLED REFRESH) IS USED TO IMPLEMENT BOTH THE DISPLAY
		14	MULTIPLEXING AND KEYBOARD SCANNING, USING THE SAME SIGNAL BOTH TO ENABLE
		15	ONE CHARACTER OF THE DISPLAY AND TO STROBE ONE ROW OF THE X-Y KEY MATRIX.
		16	THE SUBROUTINE MUST BE CALLED SUFFICIENTLY OFTEN TO ENSURE THE DISPLAY
		17	CHARACTERS DO NOT FLICKER- AT LEAST 50 COMPLETE DISPLAY SCANS PER SECOND.
		18	TO ACCOMMODATE SWITCHES OF ARBITRARY CHEAPNESS, THE DEBOUNCE TIME CAN BE
		19	SET TO BE ANY DESIRED NUMBER OF COMPLETE SCANS.
		20	THUS THE DEBOUNCE TIME IS A FUNCTION OF BOTH THE SCAN RATE AND THE VALUE
		21	OF CONSTANT 'DEBNCE'.
		22	
		23	IN THIS LISTING, THE INTERNAL TIMER IS USED TO GENERATE INTERRUPTS THAT
		24	SERVE AS A TIME BASE FOR THE REFRESH SUBROUTINE.
		25	ALTERNATE TIME BASES MIGHT BE AN EXTERNAL OSCILLATOR (DRIVING THE INTERRUPT
		26	PIN OR POLLED BY A TEST OR INPUT PIN), A SOFTWARE DELAY LOOP IN THE BACKGROUND
		27	PROGRAM, OR PERIODIC CALLS TO THE SUBROUTINE FROM THROUGHOUT THE USER'S PROGRAM
		28	AT APPROPRIATE PLACES.
		29	IN THESE CASES, THE CODE STARTING AT LABEL TIINT (TIMER INTERRUPT) AND TIRET
		30	(TIINT RETURN) COULD STILL BE USED TO SAVE AND RESTORE ACCUMULATOR CONTENTS.
		31	THE INTERRUPT SERVICING ROUTINE SELECTS REGISTER BANK 1
		32	FOR THE NEEDED REGISTERS.
		33	
		34	
		35	WRITTEN BY JOHN WHARTON, INTEL SINGLE-CHIP COMPUTER APPLICATIONS
		36	
		37	\$EJECT

LOC	OBJ	SEQ	SOURCE STATEMENT
38			; IN THIS IMPLEMENTATION OF THE DISPLAY SCAN, IT IS ASSUMED THAT THERE WILL
39			; BE RELATIVELY LITTLE I/O OTHER THAN FOR THE KEYBOARD/DISPLAY.
40			; IF THIS IS THE CASE, THEN THERE IS NO NEED FOR FOR ANY ADDITIONAL EXTERNAL
41			; LOGIC (SUCH AS ONE-OF-EIGHT DECODERS OR SEVEN-SEGMENT ENCODERS), THOUGH
42			; THERE WILL STILL BE A NEED FOR CURRENT OR VOLTAGE DRIVERS, ACCORDING TO
43			; THE TYPE OF DISPLAY BEING USED.
44			
45			; IN THIS LISTING, THE PROCESSOR I/O PORTS ARE LOGICALLY DIVIDED AS FOLLOWS:
46			
47			; PDIGIT-EIGHT BIT PORT USED TO ENABLE, ONE AT A TIME, THE INDIVIDUAL
48			; CHARACTERS OF AN EIGHT DIGIT SEVEN-SEGMENT DISPLAY, WHILE ALSO
49			; STROBING THE ROWS OF AN X-Y MATRIX KEYBOARD.
50			; BIT7 ENABLES THE LEFTMOST CHARACTER AND THE BOTTOM ROW OF THE KBD,
51			; BIT4 ENABLES THE TOP ROW OF THE 4x4 KBD AND THE FOURTH CHARACTER,
52			; BIT0 ENABLES THE RIGHTMOST CHARACTER.
53			; (A 4x8 KEYBOARD COULD BE STROBED BY ALSO USING BIT3-BIT0
54			; AND EXTENDING OR ELIMINATING THE TABLE, "LEGENDS".)
55			; THE ENABLING OF ONE BIT (ACTIVE HIGH OR LOW) IS ACCOMMODATED BY
56			; ACCESSING A LOOK-UP TABLE CALLED CHRSTB.
57			; THIS TECHNIQUE TAKES ABOUT FOUR BYTES MORE ROM THAN A TECHNIQUE
58			; OF ROTATING A 'ONE' THROUGH A FIELD OF 'ZEROS' IN THE ACC
59			; AN APPROPRIATE NUMBER OF TIMES, BUT IT ALLOWS SOME ADDITIONAL
60			; FLEXABILITY: IF THE DRIVERS BEING USED HAVE A COMBINATORIAL INPUT
61			; (AS IN THE 7545X FAMILY OF HIGH-CURRENT, HIGH-VOLTAGE DRIVERS),
62			; THE CHRSTB TABLE COULD PROVIDE ENCODED OUTPUTS. NINE DIGITS, FOR
63			; EXAMPLE, COULD BE ENABLED WITH SIX BITS OF (BUFFERED) OUTPUT:
64			; (001001, 001010, 001100, 010001, 010010, 010100, 100001, 100010, 100100)
65			; IF I/O LINES NEED TO BE CONSERVED, OR IF MANY DIGITS
66			; MUST BE DISPLAYED, AN EXTERNAL DECODER COULD BE ADDED TO THE SYSTEM.
67			; DURING CHARACTER TRANSITIONS A 'BLANK' CHARACTER IS
68			; EXPLICITLY WRITTEN TO THE DISPLAY. THUS,
69			; THERE WILL BE NO CHARACTER 'SHADOWING' CAUSED BY THE
70			; FACT THAT THE HARDWARE OR SOFTWARE DECODER KEEPS ONE
71			; OUTPUT, AND THUS ONE CHARACTER, ACTIVE AT ALL TIMES.
72			
73			; PSGMNT-EIGHT BIT PORT TO ENABLE THE SEVEN SEGMENTS & D.P. OF A STANDARD
74			; DISPLAY.
75			; BIT7-BIT0 CORRESPOND TO THE DP AND SEGMENTS G THROUGH A, RESPECTIVELY.
76			; IT IS POSSIBLE TO ACCOMMODATE
77			; DRIVERS WHICH ARE EITHER LOGICALLY INVERTING OR NON-INVERTING BY
78			; SETTING VARIABLE 'SEGPOL' (SEGMENT POLARITY).
79			; NOTE THAT BY HAVING ARBITRARY CONTROL OVER EACH SEGMENT, NON-NUMERIC
80			; CHARACTERS CAN BE REPRESENTED ON A SEVEN SEGMENT DISPLAY,
81			; AS SHOWN IN EXAMPLE SUBROUTINE 'TEST2'.
82			
83			\$EJECT

ISIS-II MCS-48/UPI-41 MACRO ASSEMBLER, V2.0
 AP40: INTEL MCS-48 KEYBOARD/DISPLAY APPLICATION NOTE APPENDIX

LOC	OBJ	SEQ	SOURCE STATEMENT
		84	; PINPUT-FOUR HIGH-ORDER BITS USED AS INPUTS FROM THE KEYBOARD RETURN LINES
		85	; ASSUMES THAT A KEY DOWN IN THE CURRENTLY ENABLED ROW WOULD RETURN
		86	A LOW LEVEL.
		87	; IN THIS CASE, BIT7 RETURNS THE LEFTMOST COLUMN, BIT4 THE RIGHTMOST.
		88	; THE HIGH-ORDER BITS ARE USED SO THAT IF AN OFF-CHIP DECODER IS USED
		89	TO ENABLE UP TO 16 CHARACTERS, FOR EXAMPLE, IT COULD BE DRIVEN BY
		90	THE LOW ORDER BITS OF THE SAME PORT.
		91	; NOTE ALSO THAT IF A SIXTEEN KEY MATRIX WERE ELECTRICALLY ORGANIZED
		92	IN A 2X8 ARRAY, ONLY TWO RETURN LINES WOULD BE NEEDED.
		93	; (IN THIS CASE, PERHAPS T0 AND T1 COULD BE USED FOR INPUT BITS.)
		94	;
		95	; PULL-UP RESISTORS ON THE RETURN LINES MIGHT BE IN ORDER IF THERE IS ANY
		96	POSSIBILITY OF A HIGH-IMPEDENCE CONDUCTIVE PATH THROUGH THE SWITCH WHEN
		97	IT IS SUPPOSED TO BE 'OPEN'.
		98	; (THIS PHENOMENON HAS ACTUALLY BEEN OBSERVED.)
		99	;
		100	; THE DRIVERS USED IN THE PROTOTYPE WERE ALL NON-INVERTING IN THAT
		101	A HIGH LEVEL ON AN OUTPUT LINE IS USED TO TURN A CHARACTER OR SEGMENT ON.
		102	THERE ARE A TOTAL OF SEVEN I/O LINES LEFT OVER.
		103	;
		104	; THE ALGORITHM FOR DRIVING THE DISPLAY USES A BLOCK OF INTERNAL RAM
		105	AS DISPLAY REGISTERS, WITH ONE BYTE CORRESPONDING TO EACH CHARACTER OF THE
		106	DISPLAY. THE EIGHT BITS OF EACH BYTE CORRESPOND TO THE SEVEN SEGMENTS & DP
		107	OF EACH CHARACTER. IF AN EXTERNAL ENCODER IS USED (SUCH AS A FOUR-BIT TO
		108	SEVEN-SEGMENT ENCODER OR A ROM FOR TRANSLATING ASCII TO
		109	SIXTEEN-SEGMENT "STARBURST" DISPLAY PATTERNS), THE TABLE ENTRIES WOULD HOLD
		110	THE CHARACTER CODES. (IN THE FORMER CASE, AN UNUSED BIT COULD BE USED TO
		111	ENABLE THE D.P.)
		112	; THUS, WRITING CHARACTERS TO THE DISPLAY FROM THE BACKGROUND PROGRAM
		113	REALLY ENTAILS WRITING THE APPROPRIATE SEGMENT
		114	PATTERNS TO A DISPLAY REGISTER- THE ACTUAL OUTPUTTING IS AUTOMATIC.
		115	THE LEFTMOST CHARACTER CORRESPONDS TO THE LAST BYTE OF THE DISPLAY
		116	REGISTERS, AND IS ACCESSED BY NEXTPL=8 (SEE SOURCE); THE RIGHTMOST
		117	CHARACTER IS THE FIRST DISPLAY BYTE, WHEN NEXTPL=1.
		118	; UTILITY SUBROUTINES ARE INCLUDED HERE TO TRANSLATE FOUR BIT NUMBERS TO HEX
		119	DIGIT PATTERNS, AND WRITE THEM INTO THE DISPLAY REGISTERS SEQUENTIALLY
		120	; (EITHER FILLING FROM THE LEFT- H.P. CALCULATOR STYLE OR FROM THE
		121	RIGHT- T.I. STYLE; SUBROUTINES WDISP AND RENTRY, RESPECTIVELY).
		122	;
		123	; THE KEYBOARD SCANNING ALGORITHM SHOWN HERE REQUIRES A KEY BE DOWN FOR
		124	SOME NUMBER OF COMPLETE DISPLAY SCANS TO BE ACKNOWLEDGED. SINCE IT IS
		125	INTENDED FOR 'ONE-FINGER' OPERATION, TWO-KEY ROLLOVER/N-KEY LOCKOUT HAS
		126	BEEN IMPLEMENTED. HOWEVER, MODIFICATIONS WOULD BE POSSIBLE TO ALLOW, FOR
		127	EXAMPLE, ONE KEY IN THE MATRIX TO BE USED AS A SHIFT KEY OR CONTROL KEY
		128	TO BE HELD DOWN WHILE ANOTHER KEY IN THE MATRIX IS PRESSED. (SEE NOTE WITHIN
		129	THE BODY OF THE LISTING.)
		130	;
		131	\$EJECT

LOC	OBJ	SEQ	SOURCE STATEMENT
132			; (BE AWARE THAT NO MORE THAN TWO KEYS CAN EVER BE DOWN UNLESS DIODES
133			; ARE PLACED IN SERIES WITH ALL OF THE SWITCHES- CERTAINLY NOT THE CASE FOR EL
134			; CHEAPO KEYBOARDS- BECAUSE SOME COMBINATIONS OF THREE KEYS DOWN WILL RESULT
135			; IN A 'PHANTOM' FOURTH KEY BEING PERCEIVED.
136			; THE PHANTOM KEY WOULD BE THE FOURTH 'CORNER' WHEN THREE KEYS FORMING
137			; A RECTANGULAR PATTERN (IN THE X-Y KEY MATRIX) ARE DOWN.)
138			; IF DIODES ARE PLACED IN THE SCANNING ARRAY, CONSIDERATIONS MUST BE MADE
139			; ABOUT HOW THE DIODE VOLTAGE DROP WILL AFFECT INPUT LOGIC LEVELS.
140			;
141			; WHEN A DEBOUNCED KEY IS DETECTED, THE NUMBER OF ITS POSITION IN THE KEY
142			; MATRIX (LEFT-TO-RIGHT, BOTTOM-TO-TOP, STARTING FROM 00) IS PLACED INTO
143			; RAM LOCATION 'KDBUF'. AN INPUT SUBROUTINE THEN NEED ONLY READ THIS LOCATION
144			; REPEATEDLY TO DETERMINE WHEN A KEY HAS BEEN PRESSED. WHEN A KEY IS DETECTED,
145			; A SPECIAL CODE BYTE SHOULD BE WRITTEN BACK TO INTO 'KDBUF' TO PREVENT
146			; REPEATED DETECTIONS OF THE SAME KEY.
147			; THE ROUTINE 'KBDIN' DEMONSTRATES A TYPICAL INPUT PROTOCOL, ALONG WITH A METHOD
148			; FOR TRANSLATING A KEY POSITION TO ITS ASSOCIATED SIGNIFICANCE BY ACCESSING
149			; TABLE 'LEGND5' IN ROM.
150			;
151			\$EJECT

LOC	OBJ	SEQ	SOURCE STATEMENT
		152	*****
		153	;
		154	INITIAL EQUATES TO DEFINE SYSTEM CONFIGURATION
		155	;
		156	*****
		157	;
0010		158	PDIGIT EQU BUS ;USED TO ENABLE CHARACTERS AND STROBE ROWS OF KEYBOARD
0008		159	PSGMNT EQU P1 ;USED TO TURN ON SEGMENTS OF CURRENTLY ENABLED DIGIT
0009		160	PINPUT EQU P2 ;PORT USED TO SCAN FOR KEY CLOSURES
		161	;
		162	;(NOTE THAT THIS PORT ALLOCATION USES THE HIGHER
		163	CURRENT SOURCING ABILITY OF THE BUS TO SWITCH ON THE
		164	DIGIT DRIVERS, AND LEAVES P23-P20 FREE FOR USING
		165	AN 8243 PORT EXPANDER IN THE SYSTEM.)
		166	POSLOG EQU 00H
0000		167	NEGLOG EQU 0FFH
		168	;
0000		169	CHARPOL EQU POSLOG ;DEFINES WHETHER OUTPUT LINES ARE ACTIVE HI OR LOW
0000		170	SEGPOL EQU POSLOG ;FOR DRIVING CHARACTERS AND SEGMENT PATTERNS
00F0		171	INPMASK EQU 0F0H ;DEFINES BITS USED AS INPUT
		172	;
0008		173	CHARNO EQU 8 ;NUMBER OF DIGITS IN DISPLAY
0004		174	NROWS EQU 4 ;ROWS OF KEYS (LESS THAN OR EQUAL TO CHARNO)
0004		175	NCOLS EQU 4 ;LESSER DIMENSION OF KEYBOARD MATRIX
		176	;
FFF0		177	TICK EQU -10H ;DETERMINES INTERRUPT INTERVAL
0004		178	DEBNCE EQU 4 ;NUMBER OF SUCCESSIVE SCANS BEFORE KEY CLOSURE ACCEPTED
0000		179	BLANK EQU 00H ;CODE TO BLANK DISPLAY CHARACTERS
		180	;
		181	;(WOULD BE 20H IF ASCII DECODING ROM USED OR 0FH IF
		182	7447-TYPE SEVEN-SEGMENT DECODER EXTERNAL TO 8748)
		183	ENCMASK EQU 0FH ;SELECTS WHICH BITS ARE RELEVANT TO ENCCAC SUBROUTINE
000F		184	;
		185	#EJECT

LOC	OBJ	SEQ	SOURCE STATEMENT	TARGET STATEMENT	LOC	OBJ
		186	*****			
		187	;			
		188	;			
		189	;			
		190	POINTERS USED FOR INDIRECT RAM ACCESSING:			
0000		191	PNTR0 EQU R0			
0001		192	PNTR1 EQU R1			
0007		193	NEXTPL EQU R7 ;USED TO KEEP TRACK OF CHARACTER POSITION BEING			
		194	;			
		195	;			
		196	*****			
		197	;			
		198	;			
		199	;			
		200	PNTR0 EQU R0 (ALREADY DEFINED)			
		201	PNTR1 EQU R1			
0002		202	ASAVE EQU R2 ;HOLDS ACCUMULATOR VALUE DURING SERVICE ROUTINE			
0004		203	ROTPAT EQU R4 ;USED TO HOLD INPUT PATTERN BEING ROTATED THROUGH CY			
0005		204	ROTCNT EQU R5 ;COUNTS NUMBER OF BITS ROTATED THROUGH CY			
0006		205	LASTKY EQU R6 ;HOLDS KEY POSITION OF LAST KEY DEPRESSION DETECTED			
0007		206	CURDIG EQU R7 ;HOLDS POSITION OF NEXT CHARACTER TO BE DISPLAYED			
		207	;			
		208	*****			
		209	;			
		210	;			
		211	DATA RAM ALLOCATION			
0020		212	NREPTS EQU 32 ;KEEPS TRACK OF SUCCESSIVE READS OF SAME KEYSTROKE			
0021		213	KEYLOC EQU 33 ;INCREMENTED AS SUCCESSIVE KEY LOCATIONS SCANNED			
0022		214	KDBBUF EQU 34 ;CARRIES POSITION OF DEBOUNCED KEY FROM REFRSH ROUTINE			
		215	;			
0023		216	RDELAY EQU 35 ;NON-ZERO WHEN DISPLAY IN PROGRESS			
		217	;			
		218	;			
		219	THE LAST <CHARNO> REGISTERS HOLD THE DISPLAY SEGMENT PATTERNS			
0037		220	SEGMAP EQU (63-CHARNO) ;BASE OF REGISTER ARRAY FOR DISPLAY PATTERNS			
		221	;			
		222	;			
		223	*****			
		224	;			
		225	;			
		226	NOTE THAT LASTKY, CURDIG, AND F1 RETAIN STATUS INFORMATION FROM			
		227	ONE INTERRUPT TO THE NEXT. ALL OTHER REGISTERS MAY BE USED IN			
		228	THE USER'S OWN INTERRUPT SERVICING ROUTINE			
		229	*****			
		230	;			
		231	\$EJECT			

LOC	OBJ	SEQ	SOURCE STATEMENT
		232 ;	
		233 ;*****	
		234 ;	
0000		235 ORG 000H	
0000 0460		236 JMP INIT	
		237 ;	
		238 ;	
		239 ;*****	
0007		240 ;	
		241 ORG 007H	
		242 ;	
		243 ;TIINT TIMER INTERRUPT SUBROUTINE	
		244 ; CALL MADE TO LOC 007H WHEN TIMER TIMES OUT.	
		245 ; TIMER CAN BE RE-INITIALIZED AT THIS POINT IF DESIRED.	
		246 ; USED HERE TO CAUSE THE DISPLAY REFRESH AND KEY SCAN ROUTINES TO	
		247 ; BE CALLED PERIODICALLY.	
0007 D5		248 TIINT: SEL R81	
0008 AA		249 MOV ASAVE, A	
0009 23F0		250 MOV A, #TICK	
000B 62		251 MOV T, A ;RELOAD TIMER INTERVAL	
		252 ;	
		253 ;*****	
		254 ;	
		255 ; THE USER'S OWN TIMER INTERRUPT ROUTINE (IF IT EXISTS) COULD	
		256 ; BE PLACED AT THIS POINT	
		257 ;	
		258 ;*****	
		259 ;	
000C 1410		260 CALL REFRESH ;CAUSE DISPLAY TO BE UPDATED	
		261 ;	
		262 ; THE COMPLETE INTERRUPT ROUTINE SHOULD BE COPIED HERE	
		263 ; TO SAVE A FULL LEVEL OF SUBROUTINE NESTING.	
		264 ; IT WAS WRITTEN AS A SUBROUTINE HERE FOR THE SAKE OF CLARITY.	
		265 ;	
		266 ;*****	
		267 ;	
		268 ;TIRET TIMER INTERRUPT RETURN CODE- RESTORES ACC VALUE	
000E FA		269 TIRET: MOV A, ASAVE	
000F 93		270 RETR	
		271 ;	
		272 \$EJECT	

LOC	OBJ	SEQ	SOURCE STATEMENT
		273	*****
		274	REFRSH SUBROUTINE TO MULTIPLEX SEVEN-SEGMENT DISPLAYS.
		275	EACH CALL CAUSES THE NEXT CHARACTER TO BE DISPLAYED,
		276	ACCORDING TO THE CONTENTS OF THE SEGMAP REGISTER ARRAY.
		277	REFRSH SHOULD BE CALLED AT LEAST EVERY MSEC OR SO.
		278	*****
		279	;
0010	2300	280	REFRSH: MOV A, #BLANK XOR SEGPOL
0012	39	281	OUTL PSGMT, A ;WRITE BLANK PATTERN TO SEG DRIVERS
0013	2357	282	REFR1: MOV A, #CHRSTB ;LOOK UP DIGIT ENABLE PATTERN
0015	6F	283	ADD A, CURDIG ;ADD CURDIG DISPLACEMENT
0016	A3	284	MOVP A, @A ;ENABLE ONE BIT OF ACCUMULATOR
0017	02	285	OUTL PDIGIT, A ;ENERGIZE CHARACTER
		286	;
		287	;
0018	2337	288	MOV A, #SEGMAP ;LOAD BASE OF REGISTER ARRAY
001A	6F	289	ADD A, CURDIG ;ADD CURDIG DISPLACEMENT
001B	A9	290	MOV PNTR1, A
001C	F1	291	MOV A, @PNTR1 ;LOAD ACC W/ NEXT SEGMENT PATTERN
001D	39	292	OUTL PSGMT, A ;ENABLE APPROPRIATE SEGMENTS
		293	;
		294	*****
		295	THE NEXT CHARACTER IS NOW BEING DISPLAYED.
		296	THE KEYBOARD SCAN ROUTINE IS INTEGRATED INTO THE DISPLAY SCAN.
		297	WITH THE CURRENT ROW ENERGIZED, CHECK IF THERE ARE ANY INPUTS.
		298	*****
		299	;
001E	B821	300	SCAN: MOV PNTR0, #KEYLOC ;SET POINTER FOR SEVERAL KEYLOC REFERENCES
0020	0A	301	IN A, PINPUT ;LOAD ANY SWITCH CLOSURES
		302	;
		303	*****
		304	## THIS BLOCK OF CODE IS NOT NEEDED BY THE KEYBOARD SCAN LOGIC. ##
		305	## HOWEVER, ITS INCLUSION WOULD SPEED THINGS UP A BIT BY ##
		306	## SKIPPING OVER ROWS IN WHICH NO KEYS ARE DOWN. ##
		307	## IT WAS OMITTED HERE TO CONSERVE ROM SPACE, BUT MIGHT BE ##
		308	## RESTORED IF VERY LARGE KEYBOARDS (ESPECIALLY THOSE WITH EIGHT ##
		309	## KEYS PER ROW) ARE TO BE USED WITH THIS ALGORITHM. ##
		310	*****
		311	## CPL A ;ANY CLOSURES DETECTED ARE NOW ONE BITS ##
		312	## ANL A, #INPMASK ##
		313	## JNZ SCAN1 ;-IF A KEY IN THE CURRENTLY ENABLED ROW IS DOWN ##
		314	## NO KEY IS NOW DOWN SO THE KEYLOC COUNT MAY BE UPDATED DIRECTLY ##
		315	## MOV A, @PNTR0 ##
		316	## ADD A, #NCOLS ##
		317	## MOV @PNTR0, A ##
		318	## JMP SCAN6 ##
		319	*****
		320	## IF THIS CODE IS USED, SUBSTITUTE THE 'JC SCAN5' FOUR LINES ##
		321	## HENCE WITH 'JNC SCAN5' TO ACCOMODATE THE INVERTED POLARITY ##
		322	*****
		323	\$EJECT

LOC	OBJ	SEQ	SOURCE STATEMENT
		324 ;	*****
		325 ;	ROTATE BITS THROUGH THE CY WHILE INCREMENTING KEYLOC.
		326 ;	*****
		327 ;	
0021	BD04	328 SCAN1:	MOV ROTCNT, #NCOLS ; SET UP FOR <NCOLS> LOOPS THROUGH 'NXTLOC'
0023	F7	329 NXTLOC:	RLC A
0024	AC	330	MOV ROTPAT, A ; SAVE SHIFTED BIT PATTERN
0025	F63F	331	JC SCAN5 ; ONE BIT IN CY INDICATES KEY NOT DOWN
		332 ;	
		333 ;	*****
		334 ;	
		335 ;	AT THIS POINT IT HAS JUST BEEN DETERMINED THAT THE VALUE
		336 ;	OF KEYLOC IS THE POSITION OF A KEY WHICH IS NOW DOWN.
		337 ;	THE FOLLOWING CODE DEBOUNCES THE KEY, ETC.
		338 ;	IF MODIFICATIONS TO THE KEYBOARD LOGIC, I.E. THE INCLUSION
		339 ;	OF A SHIFT, CONTROL, OR MODE KEY IN THE KEY MATRIX ITSELF)
		340 ;	ARE DESIRED, THEY SHOULD BE MADE AT THIS POINT, BEFORE
		341 ;	THE DEBOUNCE LOGIC BEGINS. FOR EXAMPLE, AT THIS POINT
		342 ;	KEYLOC COULD BE COMPARED AGAINST THE POSITION OF THE MODE
		343 ;	KEY, AND IF THEY MATCH SET SOME FLAG BIT AND JUMP TO
		344 ;	LABEL 'SCAN5'. OR, BY COMPARING KEYLOC AGAINST THE LAST
		345 ;	KEY DEBOUNCED, IMMEDIATE TWO-KEY ROLLOVER COULD BE
		346 ;	IMPLEMENTED.
		347 ;	
		348 ;	*****
		349 ;	
0027	A5	350	CLR F1 ; MARK THAT AT LEAST ONE KEY WAS DETECTED
0028	B5	351	CPL F1 ; \ IN THE CURRENT SCAN
		352 ;	
		353 ;	*****
		354 ;	A KEYSTROKE WAS DETECTED FOR THE CURRENT COLUMN. ITS
		355 ;	POSITION IS IN REGISTER KEYLOC. SEE IF SAME KEY SENSED LAST CYCLE.
		356 ;	*****
		357 ;	
0029	F0	358	MOV A, @PNTR0 ; PNTR0 STILL HOLDS #KEYLOC
002A	2E	359	XCH A, LASTKY
002B	DE	360	XRL A, LASTKY
002C	B820	361	MOV PNTR0, #NREPTS ; PREPARE TO CHECK AND/OR MODIFY REPEAT COUNT
002E	C634	362	JZ SCAN3
		363 ;	
		364	\$EJECT

LOC	HEX	SOURCE STATEMENT
	365 ;	*****
	366 ;	A DIFFERENT KEY WAS READ ON THIS CYCLE THAN ON THE PREVIOUS CYCLE.
	367 ;	SET NREPTS TO THE DEBOUNCE PARAMETER FOR A NEW COUNTDOWN.
	368 ;	*****
	369 ;	
0030 B004	370	MOV @PNTR0,#DEBNCE
0032 043F	371	JMP SCANS
	372 ;	
	373 ;	*****
	374 ;	SAME KEY WAS DETECTED AS ON PREVIOUS CYCLE
	375 ;	LOOK AT NREPTS: IF ALREADY ZERO, DO NOTHING.
	376 ;	ELSE DECREMENT NREPTS.
	377 ;	IF THIS RESULTS IN ZERO, MOVE LASTKY INTO KBDBUF.
	378 ;	*****
	379 ;	
0034 F0	380	SCANS: MOV A,@PNTR0
0035 C63F	381	JZ SCANS ; IF ALREADY ZERO
0037 07	382	DEC A ; INDICATE ONE MORE SUCCESSIVE KEY DETECTION
0038 A0	383	MOV @PNTR0,A
0039 963F	384	JNZ SCANS ; IF DECREMENT DOES NOT RESULT IN ZERO
003B FE	385	MOV A, LASTKY
003C B822	386	MOV PNTR0, #KBDBUF
003E A0	387	MOV @PNTR0, A ; TO MARK NEW KEY CLOSURE
	388 ;	
003F B821	389	SCANS: MOV PNTR0, #KEYLOC
0041 10	390	INC @PNTR0
0042 FC	391	MOV A, ROTPAT
0043 ED23	392	DJNZ ROTCNT, NXTLOC
	393 ;	
	394 ;	
0045 EF57	395	SCANS: DJNZ CURDIG, SCANS
	396 ;	
	397	\$EJECT

LOC	OBJ	SEQ	SOURCE STATEMENT
		398 ;	
		399 ;*****	
		400 ;	THE FOLLOWING CODE SEGMENT IS USED BY THE KEYBOARD SCANNING ROUTINE
		401 ;	IT IS EXECUTED ONLY AFTER A REFRESH SEQUENCE OF ALL
		402 ;	THE CHARACTERS IN THE DISPLAY IS COMPLETED
		403 ;*****	
		404 ;	
0047	BF08	405	MOV CURDIG, #CHARNO
0049	B000	406	MOV @PNTR0, #0 ;PNTR0 STILL CONTAINS #KEYLOC
004B	764F	407	JF1 SCAN8 ;JUMP IF ANY KEYS WERE DETECTED
004D	BEFF	408	MOV LASTKY, #0FFH ;CHANGE <LASTKY> WHEN NO KEYS ARE DOWN
004F	A5	409	SCAN8: CLR F1
		410 ;	
		411 ;*****	
		412 ;	THE NEXT CODE SEGMENT IS THE INTERRUPT-DRIVEN PORTION OF THE 'DELAY'
		413 ;	UTILITY. IT DECREMENTS RAM LOCATION 'RDELAY' ONCE PER DISPLAY SCAN
		414 ;	IF 'RDELAY' IS NOT ALREADY ZERO.
		415 ;*****	
		416 ;	
0050	B923	417	MOV PNTR1, #RDELAY
0052	F1	418	MOV A, @PNTR1
0053	C65F	419	JZ SCAN9
0055	07	420	DEC A
0056	A1	421	MOV @PNTR1, A
		422 ;	
0057	83	423	SCAN9: RET
		424 ;	
		425 ;*****	
		426 ;	
		427 ;	CHRSTB IS THE BASE FOR THE PATTERNS TO ENABLE ONE-OF-CHARNO CHARACTERS.
0057		428	CHRSTB EQU (\$-1) AND 0FFH
0058	01	429	DB (0000001B XOR CHRPOL)
0059	02	430	DB (00000010B XOR CHRPOL)
005A	04	431	DB (00000100B XOR CHRPOL)
005B	08	432	DB (00001000B XOR CHRPOL)
005C	10	433	DB (00010000B XOR CHRPOL)
005D	20	434	DB (00100000B XOR CHRPOL)
005E	40	435	DB (01000000B XOR CHRPOL)
005F	80	436	DB (10000000B XOR CHRPOL)
		437 ;	
		438	\$EJECT

LOC	OBJ	SEQ	SOURCE STATEMENT
		439	;INIT INITIALIZES PROCESSOR REGISTERS
0060	D5	440	INIT: SEL RB1
0061	BF08	441	MOV CURDIG, #CHARNO
0063	B822	442	MOV PNTR0, #KBD0BUF
0065	B0FF	443	MOV @PNTR0, #0FFH
0067	B821	444	MOV PNTR0, #KEYLOC
0069	B000	445	MOV @PNTR0, #0
006B	23F0	446	MOV A, #INPMASK
006D	3A	447	OUTL PINPUT, A ;SET BIDIRECTIONAL INPUT LINES
006E	C5	448	SEL RB0
006F	149E	449	CALL CLEAR ;UTILITY FOR SETTING INITIAL DISPLAY REGISTERS.
0071	A5	450	CLR F1
0072	23F0	451	MOV A, #TICK ;LOAD INTERRUPT RATE VALUE
0074	62	452	MOV T, A
0075	55	453	STRT T
0076	25	454	EN TCNTI ;ENABLE TIMER INTERRUPTS
		455	;
		456	;
		457	*****
		458	;
		459	ECHO CHECK FOR ANY NEW KEYSTROKES DETECTED.
		460	;
		461	TRANSLATE EACH KEYSTROKE INTO A SEGMENT PATTERN
		462	AND WRITE IT INTO THE APPROPRIATE DISPLAY REGISTER.
		463	*****
		464	;
0077	1483	465	ECHO: CALL KBDIN ;GET NEXT KEYSTROKE
0079	B281	466	JBS FKEY ;JUMP IF KEY IN RIGHTHAND COLUMN
		467	;
		468	SINCE THE ACC IS USED BY ENCACC AND RENTRY, ITS CONTENTS MUST
		469	BE PROCESSED OR SAVED BEFORE ENCACC IS CALLED
007B	148A	469	CALL ENCACC ;FORM APPROPRIATE SEGMENT PATTERN
007D	140B	470	CALL RENTRY ;WRITE PATTERN INTO DISPLAY REGISTERS
007F	0477	471	JMP ECHO ;LOOP INDEFINITELY
		472	;
0081	2400	473	FKEY: JMP FUNCTN ;JUMP TO OFF-PAGE CODE TO CALL DEMO ROUTINE
		474	;
		475	#EJECT

```

LOC OBJ      SEQ      SOURCE STATEMENT
-----
476 ;*****
477 ;
478 ; THE FOLLOWING SUBROUTINES IMPLEMENT THE UTILITIES COMMONLY USED FOR
479 ; MOST KEYBOARD/DISPLAY APPLICATIONS.
480 ; THEY COULD BE USED EXACTLY AS SHOWN HERE OR ADAPTED FOR SPECIAL CASES.
481 ;
482 ;*****
483 ;
484 ; KBDIN: KEYBOARD INPUT SUBROUTINE.
485 ; COULD BE USED TO INTERFACE THE USER'S BACKGROUND PROGRAM WITH
486 ; THE INTERRUPT DRIVEN KEYBOARD SCANNER.
487 ; RETURNS ONLY AFTER A NEW KEYSTROKE HAS BEEN DETECTED AND DEBOUNCED.
488 ; ENCODED VALUE OF KEY (RATHER THAN ITS POSITION IN SWITCH MATRIX) IS
489 ; RETURNED IN THE ACCUMULATOR.
0083 B922 490 KBDIN: MOV     PNTR1, #KDBUF
0085 2380 491     MOV     A, #00H      ; KDBUF WILL BE MARKED AS CLEAR
0087 21    492     XCH     A, @PNTR1 ; LOAD BUFFER VALUE
0088 F283 493     JB7     KBDIN
008A 038E 494     ADD     A, #LEGND5 ; ADD BASE OF KEY ENCODING TABLE
008C A3    495     MOVP    A, @A      ; OBTAIN BYTE REPRESENTING KEY SIGNIFICANCE
008D 83    496     RET
497 ;
498 ;
499 ; LEGND5 IS THE BASE FOR TABLE SHOWING KEY MATRIX SIGNIFICANCE
500 ; FOR THE KEYBOARD USED IN THE PROTOTYPE.
501 ; KEY LAYOUT IS AS SHOWN TO THE RIGHT.
502 ;
503 ; NOTE THAT BIT6-BIT4 MAY BE USED TO ENCODE KEY TYPE. IN THIS CASE:
504 ; BIT4 INDICATES REGULAR DECIMAL DIGITS,
505 ; BIT5 INDICATES RIGHT-COLUMN FUNCTION KEYS,
506 ; BIT6 INDICATES PUNCTUATION MARKS ( * AND # ).
507 ;
008E    508 LEGND5 EQU     ($ AND 0FFH) ; USE LOW ORDER BITS AS TABLE INDEX
008E 4F    509     DB      4FH
008F 10    510     DB      10H
0090 4E    511     DB      4EH
0091 28    512     DB      28H      ; PDIGIT4==> 1 2 3 <1>
0092 17    513     DB      17H
0093 18    514     DB      18H      ; PDIGIT5==> 4 5 6 <2>
0094 19    515     DB      19H
0095 24    516     DB      24H      ; PDIGIT6==> 7 8 9 <3>
0096 14    517     DB      14H
0097 15    518     DB      15H      ; PDIGIT7==> * 0 # <4>
0098 16    519     DB      16H
0099 22    520     DB      22H      ;
009A 11    521     DB      11H      ;
009B 12    522     DB      12H      ;
009C 13    523     DB      13H      ;
009D 21    524     DB      21H
525 $EJECT

```

LOC	OBJ	SEQ	SOURCE STATEMENT
		526	*****
		527	
		528	CLEAR WRITES 'BLANK' CHARACTERS INTO ALL DISPLAY REGISTERS.
		529	RETURNS WITH NEXTPL SET TO LEFTMOST CHARACTER POSITION
		530	FILL WRITES SEGMENT PATTERN NOW IN ACC INTO ALL DISPLAY REGISTERS
009E	2300	531	CLEAR: MOV A, #BLANK XOR SEGPOL
00A0	B938	532	FILL: MOV PNTR1, #SEGMAP+1
00A2	BF08	533	MOV NEXTPL, #CHARNO
00A4	A1	534	CLR1: MOV @PNTR1, A ;STORE THE BLANK CODE
00A5	19	535	INC PNTR1 ;POINT TO NEXT CHARACTER TO THE LEFT
00A6	EFA4	536	DJNZ NEXTPL, CLR1
00A8	BF08	537	MOV NEXTPL, #CHARNO
00AA	83	538	RET
		539	
		540	*****
		541	
		542	PRINT SUBROUTINE TO COPY A STRING OF BIT PATTERNS FROM ROM TO THE
		543	DISPLAY REGISTERS. STRING STARTS AT LOCATION POINTED TO BY PNTR0.
		544	CONTINUES UNTIL AN ESCAPE CODE (0FFH) IS REACHED.
		545	NOTE THAT THE CHARACTER STRING PUT OUT MUST BE LOCATED ON THE SAME
		546	PAGE AS THIS SUBROUTINE, SINCE SAME-PAGE MOVES ARE USED.
		547	PRINT IN TURN CALLS EITHER SUBROUTINE 'WDISP' OR 'RENTY'
		548	TO ACTUALLY EFFECT WRITING INTO THE DISPLAY REGISTERS.
00AB	F8	549	PRINT: MOV A, PNTR0 ;LOAD NEXT CHARACTER LOCATION
00AC	A3	550	MOVP A, @A ;LOAD BIT PATTERN INDIRECT
00AD	C6B4	551	JZ PNTR1, ;ESCAPE PATTERN
00AF	1400	552	CALL WDISP ;OUTPUT TO NEXT CHARACTER POSITION
		553	;; (CALL RENTRY INSTEAD IF MESSAGE IS TO BE RIGHT JUSTIFIED)
00B1	18	554	INC PNTR0 ;INDEX POINTER
00B2	04AB	555	JMP PRINT
00B4	83	556	PNTR1: RET ;DONE
		557	
		558	*****
		559	
		560	JOHN ARRAY HOLDS THE BIT PATTERNS FOR THE LETTERS 'JOHN' (SEE 'TEST2')
		561	(NOTE THAT 'OHN' IS WRITTEN IN LOWER CASE LETTERS)
00B5		562	JOHN EQU \$ AND 0FFH
00B5	1E	563	DB 00011110B XOR SEGPOL
00B6	5C	564	DB 01011100B XOR SEGPOL
00B7	74	565	DB 01110100B XOR SEGPOL
00B8	54	566	DB 01010100B XOR SEGPOL
00B9	00	567	DB 00
		568	
		569	\$EJECT

LOC	OBJ	SEQ	SOURCE STATEMENT
		570	*****
		571	;
		572	ENCACC ENCODES LS NIBBLE OF ACC INTO HEX BIT PATTERN INTO ACC
00BA	530F	573	ENCACC: ANL A, #ENCHSK
00BC	03C0	574	ADD A, #DGPATS
00BE	A3	575	MOVP A, @A
00BF	83	576	RET
		577	DGPATS IS THE BASE FOR THE TABLE OF SEGMENT PATTERNS FOR THE BASIC
		578	DIGITS. HERE THE FULL HEX SET (0-F) IS INCLUDED.
		579	FOR MANY USER APPLICATIONS, THE CHARACTER SET MAY BE AMENDED OR AUGMENTED
		580	TO INCLUDE ADDITIONAL SPECIAL PURPOSE PATTERNS.
		581	FORMAT IS PGFEDCBA IN STANDARD SEVEN-SEGMENT ENCODING CONVENTION
		582	WHERE P REPRESENTS THE DECIMAL POINT
00C0		583	DGPATS EQU \$ AND OFFH
00C0	3F	584	DB 00111111B XOR SEGPOL
00C1	06	585	DB 00000110B XOR SEGPOL
00C2	5B	586	DB 01011011B XOR SEGPOL
00C3	4F	587	DB 01001111B XOR SEGPOL
00C4	66	588	DB 01100110B XOR SEGPOL
00C5	6D	589	DB 01101101B XOR SEGPOL
00C6	7D	590	DB 01111101B XOR SEGPOL
00C7	07	591	DB 00000111B XOR SEGPOL
00C8	7F	592	DB 01111111B XOR SEGPOL
00C9	67	593	DB 01100111B XOR SEGPOL
00CA	77	594	DB 01110111B XOR SEGPOL
00CB	7C	595	DB 01111100B XOR SEGPOL
00CC	39	596	DB 00111001B XOR SEGPOL
00CD	5E	597	DB 01011100B XOR SEGPOL
00CE	79	598	DB 01111001B XOR SEGPOL
00CF	71	599	DB 01110001B XOR SEGPOL
		600	;
		601	*****
		602	;
		603	WDISP WRITES BIT PATTERN NOW IN ACC INTO NEXT CHARACTER POSITION
		604	OF THE DISPLAY (NEXTPL). ADJUSTS NEXTPL POINTER VALUE.
		605	RESULTS IN DISPLAY BEING FILLED LEFT TO RIGHT, THEN RESTARTING
00D0	A9	606	WDISP: MOV PNTR1, A
00D1	FF	607	MOV A, NEXTPL
00D2	0337	608	ADD A, #SEGMAP
00D4	29	609	XCH A, PNTR1
00D5	A1	610	MOV @PNTR1, A
00D6	EFDA	611	DJNZ NEXTPL, WDISP1
00D8	BF08	612	MOV NEXTPL, #CHARNO
00DA	83	613	WDISP1: RET
		614	;
		615	\$EJECT

LOC	OBJ	SEQ	SOURCE STATEMENT
		616	*****
		617	;
		618	RENTRY SUBROUTINE TO ENTER ACC CONTENTS INTO THE RIGHTMOST DIGIT
		619	AND SHIFT EVERYTHING ELSE ONE PLACE TO THE LEFT
000B	B938	620	RENTRY: MOV PNTR1, #SEGMAP+1
000D	BF08	621	MOV NEXTPL, #CHARNO
000F	21	622	RENTRY: XCH A, @PNTR1
00E0	19	623	INC PNTR1
00E1	EFDF	624	DJNZ NEXTPL, RENTRY
00E3	BF08	625	MOV NEXTPL, #CHARNO ; POINT TO LEFTMOST CHARACTER
00E5	83	626	RET
		627	;
		628	*****
		629	;
		630	RDADD TOGGLE DECIMAL POINT IN LAST CHARACTER DISPLAY CHARACTER
		631	DPADD TOGGLES DECIMAL POINT IN THE CHARACTER POINTED TO BY THE ACC.
		632	;
00E6	2301	633	RDADD: MOV A, #01H ; SET INDEX TO RIGHTMOST POSITION
00E8	0337	634	DPADD: ADD A, #SEGMAP ; ACCESS DISPLAY REGISTER FOR DESIRED PLACE
00EA	A9	635	MOV PNTR1, A
00EB	F1	636	MOV A, @PNTR1
00EC	D380	637	XRL A, #00H
00EE	A1	638	MOV @PNTR1, A
00EF	83	639	RET
		640	;
		641	*****
		642	;
		643	HOLD SUBROUTINE CALLED WHEN KEY IS KNOWN TO BE DOWN.
		644	WILL NOT RETURN UNTIL KEY IS RELEASED.
00F0	D5	645	HOLD: SEL RB1
00F1	FE	646	MOV A, LASTKY ; (LASTKY)=0FFH IFF NO KEYS DOWN
00F2	C5	647	SEL RB0
00F3	37	648	CPL A
00F4	96F0	649	JNZ HOLD
00F6	83	650	RET
		651	;
		652	*****
		653	;
		654	DELAY SUBROUTINE HANGS UP FOR THE NUMBER OF COMPLETE DISPLAY SCANS EQUAL
		655	TO THE CONTENTS OF THE ACCUMULATOR WHEN CALLED.
00F7	B923	656	DELAY: MOV PNTR1, #RDELAY
00F9	A1	657	MOV @PNTR1, A
00FA	F1	658	DELAY1: MOV A, @PNTR1
00FB	96FA	659	JNZ DELAY1
00FD	83	660	RET
		661	#EJECT

LOC	OBJ	SEQ	SOURCE STATEMENT
0100		662	ORG 100H
		663	;
		664	*****
		665	;
		666	THE CODE ON THIS PAGE IS FOR DEMONSTRATION PURPOSES ONLY-
		667	I TRULY DOUBT WHETHER ANY END USERS WOULD LIKE TO SEE A NAME
		668	POPPING UP ON THEIR CALCULATOR SCREENS.
		669	HOWEVER, THE CODE SHOWN HERE DOES INDICATE HOW THE UTILITY SUBROUTINES
		670	INCLUDED HERE COULD BE ACCESSED.
		671	THE ROUTINES THEMSELVES ARE CALLED WHEN ONE OF THE FOUR BUTTONS
		672	ON THE RIGHT-HAND SIDE OF THE PROTOTYPE KEYBOARD IS PRESSED.
		673	;
		674	*****
		675	;
		676	FUNCTN ROUTINE TO IMPLEMENT ONE OF FOUR DEMO UTILITIES, ACCORDING
		677	TO WHICH OF THE FOUR FUNCTION KEYS WAS PRESSED
0100	1212	678	FUNCTN: JB0 FUNCT1
0102	320E	679	JB1 FUNCT2
0104	520A	680	JB2 FUNCT3
		681	;
0106	14E6	682	FUNCT4: CALL RDPADD
0108	0477	683	JMP ECHO
		684	;
010A	342E	685	FUNCT3: CALL TEST3
010C	0477	686	JMP ECHO
		687	;
010E	3424	688	FUNCT2: CALL TEST2
0110	0477	689	JMP ECHO
		690	;
0112	3416	691	FUNCT1: CALL TEST1
0114	0477	692	JMP ECHO
		693	;
		694	*****
		695	;
		696	TEST1 CODE SEGMENT TO FILL DISPLAY REGISTERS WITH DIGITS DOWN TO '1'
0116	BF08	697	TEST1: MOV NEXTPL, #CHARNO
0118	B808	698	MOV PNTR0, #CHARNO ;SET FOR EIGHT LOOP REPETITIONS
011A	FF	699	TST11: MOV A, NEXTPL
011B	148A	700	CALL ENCRCC
011D	14D0	701	CALL WDISP
011F	E81A	702	DJNZ PNTR0, TST11 ;COPY NEXT DIGIT INTO DISPLAY REGISTERS
0121	BF08	703	MOV NEXTPL, #CHARNO
0123	83	704	RET
		705	;
		706	\$EJECT

LOC	OBJ	SEQ	SOURCE STATEMENT
		707	*****
		708	;
		709	TEST2 WRITES THE SEGMENT PATTERN FOR 'JOHN' ONTO THE DISPLAY,
		710	;
0124	B8B5	711	TEST2: MOV PNTR0, #JOHN
0126	14AB	712	CALL PRINT
0128	2364	713	MOV A, #100 ;SCAN DISPLAY FOR 100 CYCLES
012A	14F7	714	CALL DELAY
012C	049E	715	JMP CLEAR
		716	;
		717	*****
		718	;
		719	TEST3 SUBROUTINE TO FILL DISPLAY WITH DASHES
		720	;
		721	JUMPS INTO SUBROUTINE 'CLEAR'
		722	AS SOON AS THE KEY IS RELEASED.
012E	2340	723	TEST3: MOV A, #01000000B XOR SEGPOL ;PATTERN FOR '-'
0130	14A0	724	CALL FILL
0132	14F0	725	CALL HOLD
0134	049E	726	JMP CLEAR
		727	;
		728	*****
		729	END

USER SYMBOLS															
ASAVE	0002	BLANK	0000	CHARNO	0008	CHARPOL	0000	CHRSTB	0057	CLEAR	009E	CLR1	00A4	CURDIG	0007
DEBNCE	0004	DELAY	00F7	DELAY1	00FA	DGPATS	00C0	DPADD	00E8	ECHO	0077	ENCACC	00BA	ENCMCK	000F
FILL	00A0	FKEY	0081	FUNCT1	0112	FUNCT2	010E	FUNCT3	010A	FUNCT4	0106	FUNCTN	0100	HOLD	00F0
INIT	0060	INPMASK	00F0	JOHN	0085	KBD0BUF	0022	KBDIN	0083	KEYLOC	0021	LASTKY	0006	LEGND5	008E
NCOLS	0004	NEGLOG	00FF	NEXTPL	0007	NREPTS	0020	NROWS	0004	NXTLOC	0023	PDIGIT	0010	PINPUT	0009
PNTR0	0000	PNTR1	0001	POSLOG	0000	PRINT	00AB	PRNT1	00B4	PSGMNT	0008	RDELAY	0023	RDPADD	00E6
REFR1	0013	REFRSH	0010	RENT1	00DF	RENTY	00DB	ROTCNT	0005	ROTPAT	0004	SCAN	001E	SCAN1	0021
SCAN3	0034	SCAN5	003F	SCAN6	0045	SCAN8	004F	SCAN9	0057	SEGMAP	0037	SEGPOL	0000	TEST1	0116
TEST2	0124	TEST3	012E	TICK	FFF0	TIINT	0007	TIRET	000E	TS11	011A	WDISP	0000	WDISP1	0000

4-3	Introduction
4-3	Full Duplex Serial Communications
4-12	Multiply Algorithms
4-16	Divide Algorithms
4-18	Binary & BCD Conversions
4-24	Conclusion

Application Techniques for the 8049 Microcomputer

January 1979

Serial I/O and Math Utilities for the 8049 Microcomputer

Lionel Smith and Cecil Moore
Microcomputer Applications

Application Techniques for the 8049 Microcomputer

Contents

Introduction	4-3
Full Duplex Serial Communications	4-3
Multiply Algorithms	4-12
Divide Algorithms	4-15
Binary & B C D Conversions	4-18
Conclusion	4-24

INTRODUCTION

The Intel® MCS-48 family of microcomputers marked the first time an eight bit computer with program storage, data storage, and I/O facilities was available on a single LSI chip. The performance of the initial processors in the family (the 8748 and the 8048) has been shown to meet or exceed the requirements of most current applications of microcomputers. A new member of the family, however, has been recently introduced which promises to allow the use of the single chip microcomputer in many application areas which have previously required a multichip solution. The Intel® 8049 virtually doubles processing power available to the systems designer. Program storage has been increased from 1K bytes to 2K bytes, data storage has been increased from 64 bytes to 128 bytes, and processing speed has been increased by over 80%. (The 2.5 microsecond instruction cycle of the first members of the family has been reduced to 1.36 microseconds.)

It is obvious that this increase in performance is going to result in far more ambitious programs being written for execution in a single chip microcomputer. This article will show how several program modules can be designed using the 8049. These modules were chosen to illustrate the capability of the 8049 in frequently encountered design situations. The modules included are full duplex serial I/O, binary multiply and divide routines, binary to BCD conversions, and BCD to binary conversion. It should be noted that since the 8049 is totally software compatible with the 8748 and 8048 these routines will also be useful directly on these processors. In addition the algorithms for these programs are expressed in a program design language format which should allow them to be easily understood and extended to suit individual applications with minimal problems.

FULL DUPLEX SERIAL COMMUNICATIONS

Serial communications have always been an important facet in the application of microprocessors. Although this has been partially due to the necessity of connecting a terminal to the microprocessor based system for program generation and debug, the main impetus has been the simple fact that a large share of microprocessors find their way into end products (such as intelligent terminals) which themselves depend on serial communication. When it is necessary to add a serial link to a microprocessor such as the Intel® MCS-85 or 86 the solution is easy; the Intel® 8251A USART or 8273 SDLC chip can easily be added to provide the necessary protocol. When it is necessary to do the same thing to a single chip microcomputer, however, the situation becomes more difficult.

Some microcomputers, such as the Intel 8048 and 8049 have a complete bus interface built into them which allows the simple connection of a USART to the processor chip. Most other single chip microcomputers, although lacking such a bus, can be connected to a USART with various artificial hardware and software constructs. The difficulty with using these chips,

however, is more economic than technical; these same peripheral chips which are such a bargain when coupled to a microprocessor such as the MCS-85 or 86, have a significant cost impact on a single chip microcomputer based system. The high speed of the 8049, however, makes it feasible to implement a serial link under software control with no hardware requirements beyond two of the I/O pins already resident on the microcomputer.

There are many techniques for implementing serial I/O under software control. The application note "Application Techniques for the MCS-48 Family" describes several alternatives suitable for half duplex operation. Full duplex operation is more difficult, however, since it requires the receive and transmit processes to operate concurrently. This difficulty is made more severe if it is necessary for some other process to also operate while serial communication is occurring. Scanning a keyboard and display, for example, is a common operation of single chip microcomputer based system which might have to occur concurrently with the serial receive/transmit process. The next section will describe an algorithm which implements full duplex serial communication to occur concurrently with other tasks. The design goal was to allow 2400 baud, full duplex, serial communication while utilizing no more than 50% of the available processing power of the high speed 8049 microcomputer.

The format used for most asynchronous communication is shown in Figure 1. It consists of eight data bits with a leading 'START' bit and one or more trailing 'STOP' bits. The START bit is used to establish synchronization between the receiver and transmitter. The STOP bits ensure that the receiver will be ready to synchronize itself when the next start bit occurs. Two stop bits are normally used for 110 baud communication and one stop bit for higher rates.

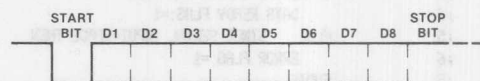


Figure 1.

The algorithm used for reception of the serial data is shown in Figure 2. It uses the on board timer of the 8049 to establish a sampling period of four times the desired baud rates. For 2400 baud operation a crystal frequency of 9.216 MHz was chosen after the following calculation:

$$f = 480N(2400)(4)$$

where 480 is the factor by which the crystal frequency is divided within the processor to get the basic interrupt rate

2400 is the desired baud rate

4 is the required number of samples per bit time

N is the value loaded into the MCS-48 timer when it overflows

The value N was chosen to be two (resulting in $f = 9.216$ MHz) so that the operating frequency of the 8049 could be as high as possible without exceeding the maximum frequency specification of the 8049 (11 MHz).

```

; START OF RECEIVE ROUTINE
; =====
;
; 1 IF RECEIVE FLAG=0 THEN
; 2   IF SERIAL INPUT=SPACE THEN
; 3     RECEIVE FLAG:=1
; 3     BYTE FINISHED FLAG:=0
; 2   ENDIF
; 1 ELSE SINCE RECEIVE FLAG=1 THEN
; 2   IF SYNC FLAG=0 THEN
; 3     IF SERIAL INPUT=SPACE THEN
; 4       SYNC FLAG:=1
; 4       DATA:=80H
; 4       SAMPLE CNTR:=4
; 3     ELSE SINCE SERIAL INPUT=MARK THEN
; 4       RECEIVE FLAG:=0
; 3     ENDIF
; 2   ELSE SINCE SYNC FLAG=1 THEN
; 3     SAMPLE COUNTER:=SAMPLE COUNTER-1
; 3     IF SAMPLE COUNTER=0 THEN
; 4       SAMPLE COUNTER:=4
; 4       IF BYTE FINISHED FLAG=0 THEN
; 5         CARRY:=SERIAL INPUT
; 5         SHIFT DATA RIGHT WITH CARRY
; 5         IF CARRY=1 THEN
; 6           OKDATA:=DATA
; 6           IF DATA READY FLAG=0 THEN
; 7             BYTE FINISHED FLAG:=1
; 6           ELSE
; 7             BYTE FINISHED FLAG:=1
; 7             OVERRUN FLAG:=1
; 6           ENDIF
; 5         ENDIF
; 4       ELSE SINCE BYTE FINISHED FLAG=1 THEN
; 5         IF SERIAL INPUT=MARK THEN
; 6           DATA READY FLAG:=1
; 5         ELSE SINCE SERIAL INPUT=SPACE THEN
; 6           ERROR FLAG:=1
; 5         ENDIF
; 5         RECEIVE FLAG:=0
; 5         SYNC FLAG:=0
; 4       ENDIF
; 3     ENDIF
; 2   ENDIF
; 1 ENDIF

```

Figure 2

The timer interrupt service routine always loads the timer with a constant value. In effect the timer is used to generate an independent time base of four times the required baud rate. This time base is free running and is never modified by either the receive or transmit programs, thus allowing both of them to use the same timer. Routines which do other time dependent tasks (such as scanning keyboards) can also be called periodically at some fixed multiple of this basic time unit.

The algorithm shown in Figure 2 uses this basic clock plus a handful of flags to process the serial input data.

Once the meaning of these flags are understood the operation of the algorithm should be clear. The **Receive Flag** is set whenever the program is in the process of receiving a character. The **Sync Flag** is set when the center of the start bit has been checked and found to be a SPACE (if a MARK is detected at this point the receiver process has been triggered by a noise pulse so the program clears the **Receive Flag** and returns to the idle state). When the program detects synchronization it loads the variable **DATA** with 80H and starts sampling the serial line every four counts. As the data is received it is right shifted into variable **DATA**; after eight bits have been received the initial one set into **DATA** will result in a carry out and the program knows that it has received all eight bits. At this point it will transfer all eight bits to the variable **OKDATA** and set the **Byte Finished Flag** so that on the next sample it will test for a valid stop bit instead of shifting in data. If this test is successful the **Data Ready Flag** will be set to indicate that the data is available to the main process. If the test is unsuccessful the **Error Flag** will be set.

The transmit algorithm is shown in Figure 3. It is executed immediately following the receive process. It is a simple program which divides the free running clock down and transmits a bit every fourth clock. The variable **TICK COUNTER** is used to do the division. The **Transmitting Flag** indicates when a character transmission is in progress and is also used to determine when the START bit should be sent. The **TICK COUNTER** is used to determine when to send the next bit (**TICK COUNTER MODULO 4 = 0**) and also when the STOP bits should be sent (**TICK COUNTER = 9 - 4**). After the transmit routine completes any other timer based routines, such as a keyboard/display scanner or a real time clock, can be executed.

```

; START OF TRANSMIT ROUTINE
; =====
;
; 1
; 1 TICK COUNTER:=TICK COUNTER+1
; 1 IF TICK COUNTER MOD 4=0 THEN
; 2   IF TRANSMITTING FLAG=1 THEN
; 3     IF TICK COUNTER=00 1010 00 BINARY THEN
; 4       TRANSMITTING FLAG:=0
; 3     ELSE IF TICK COUNTER=00 1001 00 BINARY THEN
; 4       SEND END MARK
; 4       TRANSMITTING FLAG:=0
; 3     ELSE SINCE TICK COUNTER>THE ABOVE COUNT THEN
; 4       SEND NEXT BIT
; 3     ENDIF
; 2   ELSE SINCE TRANSMITTING FLAG=0 THEN
; 3     IF TRANSMIT REQUEST FLAG=1 THEN
; 4       NEXTBYT:=NEXTBYT
; 4       TRANSMIT REQUEST FLAG:=0
; 4       TRANSMITTING FLAG:=1
; 4       TICK COUNTER:=0
; 4       SEND SYNC BIT (SPACE)
; 3     ENDIF
; 2   ENDIF
; 1 ENDIF

```

Figure 3

Figure 4 shows the complete receive and transmit programs as they are implemented in the instruction set of

the 8049. Also included in Fig. 4 is a short routine which was used to test the algorithm.

ISIS-II MCS-48/UPI-41 MACRO ASSEMBLER, V2.0

LOC	OBJ	SEQ	SOURCE STATEMENT
		1	*****
		2	;
		3	;
		4	THIS PROGRAM TESTS THE FULL DUPLEX COMMUNICATION SOFTWARE
		5	;
		6	*****
		7	\$INCLUDE(.F1:URTEST.PDL)
		8	;
		9	START OF TEST ROUTINE
		10	=====
		11	;
		12	;
		13	;
		14	;
		15	;
		16	1 ERROR COUNT:=0
		17	1 REPEAT
		18	2 PATTERN:=0
		19	2 INITIALIZE TIMER
		20	2 CLEAR FLAGBYTE
		21	2 FLAG1=MARK
		22	2 REPEAT
		23	3 IF TRANSMIT REQUEST FLAG=0 THEN
		24	4 NXTBYTE:=PATTERN
		25	4 TRANSMIT REQUEST FLAG=1
		26	3 ENDF
		27	3 IF DATA READY FLAG=1 THEN
		28	4 PATTERN:=OKDATA
		29	4 DATA READY FLAG:=0
		30	3 ENDF
		31	2 UNTIL ERROR FLAG OR OVERRUN FLAG
		32	2 INCREMENT ERROR COUNT
		33	1 UNTIL FOREVER
		34	EOF
		35	\$EJECT
0000		36	ORG 0
		37	1 SELECT REGISTER BANK 0
0000 C5		38	SEL R00
		39	1 GOTO TEST
0001 2400		40	JMP TEST
		41	\$ INCLUDE(.F1:UART)
		42	;
		43	;
		44	ASYNCHRONOUS RECEIVE/TRANSMIT ROUTINE
		45	=====
		46	THIS ROUTINE RECEIVES SERIAL CODE USING PIN T0 AS RXD
		47	AND CONCURRENTLY TRANSMITS USING PIN P27
		48	NOTE:
		49	THIS ROUTINE USES FLAG 1 TO BUFFER THE TRANSMITTED

Figure 4

```

= 51 ; 1 WOULD BE CAUSED BY VARIATIONS IN THE RECEIVE
= 52 ; 1 TIMING. NO OTHER PROGRAM MAY USE FLAG 1 WHILE
= 53 ; 1 THE TIMER INTERRUPT IS ENABLED.
= 54 ;
= 55 ;
= 56 ;
= 57 ;
= 58 ;
= 59 ; REGISTER ASSIGNMENTS-BANK1
= 60 ; =====
= 61 ;
= 62 ;
0007 = 63 ATEMP EQU R7 ; USED TO SAVE ACCUMULATOR CONTENTS DURING INTERRUPT
0006 = 64 FLGBYT EQU R6 ; CONTAINS VARIOUS FLAGS USED TO CONTROL THE RECEIVE
= 65 ; AND TRANSMIT PROCESS. SEE CONSTANT DEFINITIONS FOR
= 66 ; THE MEANING OF EACH BIT
0005 = 67 SAMCTR EQU R5 ; SAMPLE COUNTER FOR THE RECIEVE PROCESS
0004 = 68 TCKCTR EQU R4 ; SAMPLE COUNTER FOR THE TRANSMIT PROCESS
0000 = 69 REG0 EQU R0 ; USED AS POINTER REGISTER
= 70 ;
= 71 ; RAM ASSIGNMENTS
= 72 ; =====
= 73 ;
0020 = 74 MOKDAT EQU 20H ; RECEIVE RETURNS VALID DATA IN THIS BYTE
0021 = 75 MDATA EQU 21H ; RECEIVE ACCUMULATES DATA IN THIS BYTE
0022 = 76 MXMTBY EQU 22H ; CONTAINS BYTE BEING TRANSMITTED
0023 = 77 MNXTBY EQU 23H ; CONTAINS THE NEXT BYTE TO BE TRANSMITTED
= 78 $EJECT
= 79 ;
= 80 ;
= 81 ; CONSTANTS
= 82 ; =====
= 83 ;
= 84 ; THE FOLLOWING CONSTANTS ARE USED TO ACCESS THE FLAG BITS CONTAINED
= 85 ; IN REGISTER FLGBYT
= 86 ;
0001 = 87 RCVFLG EQU 01H ; SET WHEN START BIT IS FIRST DETECTED
= 88 ; RESET WHEN RECEIVE PROCESS IS COMPLETE
0002 = 89 SYNFLG EQU 02H ; SET WHEN START BIT IS VERIFIED
= 90 ; RESET WHEN RECEIVE PROCESS IS COMPLETE
0004 = 91 BYFNFL EQU 04H ; RESET WHEN START BIT IS FIRST DETECTED
= 92 ; SET WHEN THE EIGHT DATA BITS HAVE ALL BEEN RECEIVED
0008 = 93 DRDYFL EQU 08H ; SHOULD BE RESET BY MAIN PROGRAM WHEN DATA IS ACCEPTED
= 94 ; SET BY RECEIVE PROCESS WHEN STOP BIT(S) ARE VERIFIED
0010 = 95 ERRFLG EQU 10H ; SHOULD BE RESET BY MAIN PROGRAM WHEN SAMPLED
= 96 ; SET BY RECEIVE PROCESS IF A FRAMING ERROR IS DETECTED
0020 = 97 TRROFL EQU 20H ; TESTED BY MAIN PROGRAM TO DETERMINE IF READY TO
= 98 ; TRANSMIT A NEW BYTE-SET TO INDICATE THAT NXTBYT
= 99 ; HAS BEEN LOADED
= 100 ; RESET BY TRANSMIT PROCESS WHEN BYTE IS ACCEPTED
0040 = 101 TRNGFL EQU 40H ; SET WHEN TRANSMISSION OF A BYTE STARTS
= 102 ; RESET WHEN STOP BIT IS TRANSMITTED
0080 = 103 OVRUN EQU 80H ; SET BY RECEIVE PROCESS WHEN OVERUN OCCURRS
= 104 ; SHOULD BE RESET BY MAIN PROGRAM WHEN SAMPLED

```

Figure 4 (continued)

LOC	OBJ	SEQ	SOURCE STATEMENT	ADDRESS	HEX	DISASSEMBLY
		= 105 ;		00000	0000	
		= 106 ;	GENERAL CONSTANTS			
		= 107 ;	=====			
		= 108 ;				
0080		= 109 MARK	EQU 80H ; USED TO GENERATE A MARK			
FF7F		= 110 SPACE	EQU NOT 80H ; USED TO GENERATE A SPACE			
0000		= 111 STPBTS	EQU 0 ; CONTROLS THE NUMBER OF STOP BITS			
		= 112	0 GENERATES ONE STOP BIT			
		= 113	1 GENERATES TWO STOP BITS			
		= 114 ;				
		= 115 \$EJECT				
		= 116 ;				
		= 117 ;	START OF RECEIVE/TRANSMIT INTERRUPT SERVICE ROUTINE			
		= 118 ;	=====			
		= 119 ;				
0007		= 120	ORG 0007H			
		= 121				
		= 122 ;1	ENTER INTERRUPT MODE			
0007 160A		= 123 TISR:	JTF UART			
0009 93		= 124	RETR			
000A D5		= 125 UART:	SEL RB1			
		= 126 ;1	SAVE ACCUMULATOR CONTENTS			
000B AF		= 127	MOV ATEMP, A			
		= 128 ;1	RELOAD TIMER			
000C 23FE		= 129	MOV A, #TIMCNT			
000E 62		= 130	MOV T, A			
		= 131 ;				
		= 132 ;	OUTPUT TXD BUFFER (F1) TO TXD I/O LINE (P27)			
		= 133 ;	=====			
		= 134 ;				
000F 7615		= 135	JF1 OMARK			
0011 9A7F		= 136 OSPACE:	ANL P2, #SPACE			
0013 0417		= 137	JMP RCV000			
0015 8A80		= 138 OMARK:	ORL P2, #MARK			
		= 139 ;				
		= 140 ;	START OF RECEIVE ROUTINE			
		= 141 ;	=====			
		= 142 ;				
		= 143 ;1	IF RECEIVE FLAG=0 THEN			
0017 FE		= 144 RCV000:	MOV A, FLGBYT			
0018 1224		= 145	JB0 RCV010			
		= 146 ;2	IF SERIAL INPUT=SPACE THEN			
001A 3664		= 147	JT0 XMIT			
		= 148 ;3	RECEIVE FLAG:=1			
001C FE		= 149	MOV A, FLGBYT			
001D 4301		= 150	ORL A, #RCVFLG			
		= 151 ;3	BYTE FINISHED FLAG:=0			
001F 53FB		= 152	ANL A, #NOT BYFNFL			
		= 153 ;2	ENDIF			
0021 AE		= 154	MOV FLGBYT, A			
0022 0464		= 155	JMP XMIT			
		= 156 ;1	ELSE SINCE RECEIVE FLAG=1 THEN			
		= 157 ;2	IF SYNC FLAG=0 THEN			
0024 3238		= 158 RCV010:	JB1 RCV030			
		= 159 ;3	IF SERIAL INPUT=SPACE THEN			

Figure 4 (continued)

LOC	OBJ	SEQ	SOURCE STATEMENT
0026	3633	= 160	JT0 RCV020
		= 161 ; 4	SYNC FLAG:=1
0028	4302	= 162	ORL A, #SYNFLAG
002A	AE	= 163	MOV FLGBYT, A
		= 164 ; 4	DATA:=80H
002B	B821	= 165	MOV R0, #MDATA
002D	B080	= 166	MOV @R0, #80H
		= 167 ; 4	SAMPLE CNTR:=4
002F	BD04	= 168	MOV SAMCTR, #4
0031	0464	= 169	JMP XMIT
		= 170 ; 3	ELSE SINCE SERIAL INPUT=MARK THEN
		= 171 ; 4	RECEIVE FLAG:=0
0033	53FE	= 172 RCV020:	ANL A, #NOT RCVFLAG
		= 173 ; 3	ENDIF
0035	AE	= 174	MOV FLGBYT, A
0036	0464	= 175	JMP XMIT
		= 176 ; 2	ELSE SINCE SYNC FLAG=1 THEN
		= 177 ; 3	SAMPLE COUNTER:=SAMPLE COUNTER-1
0038	ED64	= 178 RCV030:	DJNZ SAMCTR, XMIT
		= 179 ; 3	IF SAMPLE COUNTER=0 THEN
		= 180 ; 4	SAMPLE COUNTER:=4
003A	BD04	= 181	MOV SAMCTR, #4
		= 182 ; 4	IF BYTE FINISHED FLAG=0 THEN
003C	5259	= 183	JB2 RCV050
003E	97	= 184	CLR CARRY
		= 185 ; 5	CARRY:=SERIAL INPUT
003F	2642	= 186	JNT0 RCV040
0041	A7	= 187	CPL C
0042	B821	= 188 RCV040:	MOV R0, #MDATA
0044	F0	= 189	MOV A, @R0
		= 190 ; 5	SHIFT DATA RIGHT WITH CARRY
0045	67	= 191	RRC A
0046	A0	= 192	MOV @R0, A
		= 193 ; 5	IF CARRY=1 THEN
0047	E664	= 194	JNC XMIT
		= 195 ; 6	OKDATA:=DATA
0049	B820	= 196	MOV R0, #OKDATA
004B	A0	= 197	MOV @R0, A
		= 198 ; 6	IF DATA READY FLAG=0 THEN
004C	FE	= 199	MOV A, FLGBYT
004D	7254	= 200	JB3 RCV045
		= 201 ; 7	BYTE FINISHED FLAG=1
004F	4304	= 202	ORL A, #BYFNFL
0051	AE	= 203	MOV FLGBYT, A
0052	0464	= 204	JMP XMIT
		= 205 ; 6	ELSE
		= 206 ; 7	BYTE FINISHED FLAG:=1
		= 207 ; 7	OVERRUN FLAG:=1
		= 208 RCV045:	
		= 209	MOV A, FLGBYT
0054	4304	= 210	ORL A, #(<BYFNFL OR OVRUN)
0056	AE	= 211	MOV FLGBYT, A
		= 212 ; 6	ENDIF
		= 213 ; 5	ENDIF
0057	0464	= 214	JMP XMIT

Figure 4 (continued)

LOC	OBJ	SEQ	SOURCE STATEMENT
		= 215 ; 4	ELSE SINCE BYTE FINISHED FLAG=1 THEN
		= 216 ; 5	IF SERIAL INPUT=MARK THEN
0059	265F	= 217 RCV050: JNT0	RCV060
		= 218 ; 6	DATA READY FLAG:=1
005B	4308	= 219	ORL A, #DRDYFL
005D	0461	= 220	JMP RCV070
		= 221 ; 5	ELSE SINCE SERIAL INPUT=SPACE THEN
		= 222 ; 6	ERROR FLAG:=1
005F	4310	= 223 RCV060: ORL	A, #ERRFLG
		= 224 ; 5	ENDIF
		= 225 ; 5	RECEIVE FLAG:=0
		= 226 ; 5	SYNC FLAG:=0
0061	53FC	= 227 RCV070: ANL	A, #NOT(SYNFLG OR RCVFLG)
0063	AE	= 228	MOV FLGBYT, A
		= 229 ; 4	ENDIF
		= 230 ; 3	ENDIF
		= 231 ; 2	ENDIF
		= 232 ; 1	ENDIF
		= 233 #EJECT	
		= 234 ;	
		= 235 ;	START OF TRANSMIT ROUTINE
		= 236 ;	=====
		= 237 ;	
		= 238 ; 1	
		= 239	; TRANSMITTER OUTPUT BIT IS P2-7
		= 240 ; 1	TICK COUNTER:=TICK COUNTER+1
0064	1C	= 241 XMIT: INC	TCKCTR
		= 242 ; 1	IF TICK COUNTER MOD 4=0 THEN
0065	2303	= 243	MOV A, #03H
0067	5C	= 244	ANL A, TCKCTR
0068	9697	= 245	JNZ RETURN
		= 246 ; 2	IF TRANSMITTING FLAG=1 THEN
006A	FE	= 247	MOV A, FLGBYT
006B	37	= 248	CPL A
006C	D206	= 249	JB6 XMT040
		= 250	IF STPBTS EQ 1
		= 251 ; 3	IF TICK COUNTER=00 1010 00 BINARY THEN
		= 252	MOV A, #20H ; CONDITIONAL ASSEMBLY
		= 253	XRL A, TCKCTR
		= 254	JNZ XMT010 ;
		= 255 ; 4	TRANSMITTING FLAG:=0
		= 256	MOV A, FLGBYT ;
		= 257	ANL A, #NOT TRNGFL ;
		= 258	MOV FLGBYT, A ;
		= 259	JMP RETURN ;
		= 260	ENDIF
		= 261 ; 3	ELSE IF TICK COUNTER=00 1001 00 BINARY THEN
006E	2324	= 262 XMT010: MOV	A, #24H
0070	DC	= 263	XRL A, TCKCTR
0071	967B	= 264	JNZ XMT020
		= 265 ; 4	SEND END MARK
0073	A5	= 266	CLR F1 ; SET FLAG1 TO MARK
0074	B5	= 267	CPL F1
		= 268	IF STPBTS EQ 0
		= 269 ; 4	TRANSMITTING FLAG:=0

Figure 4 (continued)

LOC	OBJ	SEQ	SOURCE STATEMENT
0075	FE	= 270	MOV A, FLGBYT ; CONDITIONAL ASSEMBLY
0076	53BF	= 271	ANL A, #NOT TRNGFL ;
0078	AE	= 272	MOV FLGBYT, A ;
0079	0497	= 273	JMP RETURN ;
		= 274	ENDIF
		= 275 ; 3	ELSE SINCE TICK COUNTER > THE ABOVE COUNT THEN
		= 276 ; 4	SEND NEXT BIT
007B	B822	= 277 XMT020:	MOV R0, #XMTBY
007D	F0	= 278	MOV A, @R0
007E	67	= 279	RRC A
007F	A0	= 280	MOV @R0, A
0080	A5	= 281	CLR F1 ; FLAG 1 WILL BE USED TO BUFFER TXD
0081	E697	= 282	JNC RETURN ; GO TO RETURN POINT IF TXD=SPACE (0)
0083	B5	= 283	CPL F1 ; ELSE COMPLEMENT FLAG 1 TO A MARK
0084	0497	= 284	JMP RETURN
		= 285 ; 3	ENDIF
		= 286 ; 2	ELSE SINCE TRANSMITTING FLAG=0 THEN
		= 287 ; 3	IF TRANSMIT REQUEST FLAG=1 THEN
0086	B297	= 288 XMT040:	JBS RETURN ; FLAG BYTE THERE
		= 289 ; 4	XMTBYT:=NXTBYT
0088	B823	= 290	MOV R0, #NXTBYT
008A	F0	= 291	MOV A, @R0
008B	B822	= 292	MOV R0, #XMTBYT
008D	A0	= 293	MOV @R0, A
		= 294 ; 4	TRANSMIT REQUEST FLAG:=0
008E	FE	= 295	MOV A, FLGBYT
008F	53DF	= 296	ANL A, #NOT TRNGFL
		= 297 ; 4	TRANSMITTING FLAG:=1
0091	4340	= 298	ORL A, #TRNGFL
0093	AE	= 299	MOV FLGBYT, A
		= 300 ; 4	TICK COUNTER:=0
0094	BC00	= 301	MOV TCKCTR, #0
		= 302 ; 4	SEND SYNC BIT (SPACE)
0096	A5	= 303	CLR F1 ; SET FLAG 1 TO CAUSE A SPACE
		= 304 ; 3	ENDIF
		= 305 ; 2	ENDIF
		= 306 ; 1	ENDIF
		= 307	RETURN:
		= 308 ; 1	RESTORE ACCUMULATOR
0097	FF	= 309	MOV A, ATEMP
0098	93	= 310	RETR
		311	\$EJECT
		312 ;	
		313 ;	START OF TEST ROUTINE
		314 ;	=====
		315 ;	
0100		316	ORG 0100H
FFFE		317	TIMCNT EQU -2
001E		318	NFLGBY EQU 1EH
001D		319	MSAMCT EQU 1DH
001C		320	MTCKCT EQU 1CH
		321	
0007		322	ERRCNT EQU R7
0006		323	PATT EQU R6
		324 ;	

Figure 4 (continued)

LUC	OBJ	SEQ	SOURCE STATEMENT	THATITAG2 33K002	322	140	100
		325 ;					
		326 ;					
		327 ;1 ERROR COUNT:=0					
0100	BF00	328 TEST: MOV ERRCNT, #0					
		329 ;1 REPEAT					
		330 TLOP:					
		331 ;2 PATTERN:=0					
0102	BE00	332 MOV PATT, #00					
		333 ;2 INITIALIZE TIMER					
0104	23FE	334 MOV A, #TIMCNT					
0106	62	335 MOV T, A					
0107	55	336 STRT T					
0108	25	337 EN TCNTI					
		338 ;2 CLEAR FLAGBYTE					
0109	B81E	339 MOV R0, #MFLGBY					
010B	B000	340 MOV @R0, #0					
		341 ;2 FLAG1=MARK					
010D	A5	342 CLR F1					
010E	B5	343 CPL F1					
		344 ;2 REPEAT					
		345 TILOP:					
		346 ;3 IF TRANSMIT REQUEST FLAG=0 THEN					
010F	B81E	347 MOV R0, #MFLGBY					
0111	F0	348 MOV A, @R0					
0112	B224	349 JB5 TREC					
		350 ;4 NXTBYTE:=PATTERN					
0114	B923	351 MOV R1, #NXTBY					
0116	FE	352 MOV A, PATT					
0117	A1	353 MOV @R1, A					
		354 ;4 TRANSMIT REQUEST FLAG=1					
0118	35	355 DIS TCNTI ; LOCK OUT TIMER INTERRUPT					
		356 ; SO THAT MUTUAL EXCLUSION IS MAINTAINED WHILE					
		357 ; THE FLAG BYTE IS BEING MODIFIED					
0119	F0	358 MOV A, @R0					
011A	4320	359 ORL A, #TRRGFL					
011C	A0	360 MOV @R0, A					
011D	25	361 EN TCNTI					
011E	1622	362 JTF TESTA					
0120	2424	363 JMP TREC					
0122	140A	364 TESTA: CALL UART ; CALL UART BECAUSE TIMER OVERFLOWED DURING LOCKOUT					
		365 ;3 ENDIF					
		366 ;3 IF DATA READY FLAG=1 THEN					
		367 TREC:					
0124	F0	368 MOV A, @R0					
0125	37	369 CPL A					
0126	7238	370 JB3 TRECE					
		371 ;4 PATTERN:=OKDATA					
0128	B920	372 MOV R1, #OKDAT					
012A	F1	373 MOV A, @R1					
012B	AE	374 MOV PATT, A					
		375 ;4 DATA READY FLAG:=0					
012C	35	376 DIS TCNTI ; LOCK OUT TIMER INTERRUPT					
		377 ; SO THAT MUTUAL EXCLUSION IS MAINTAINED WHILE					
		378 ; THE FLAG BYTE IS BEING MODIFIED					
012D	F0	379 MOV A, @R0					

Figure 4 (continued)

```

0130 00      381      MOV     @R0, A
0131 25      382      EN      TCNTI
0132 1636    383      JTF     TESTB
0134 2438    384      JMP     TRECE
0136 140A    385 TESTB: CALL    UART      ; CALL UART IF TIMER OVERFLOWED DURING LOCKOUT
0136 140A    386 TRECE:
0136 140A    387 ;3      ENDIF
0138 F0      388 ;2      UNTIL ERROR FLAG OR OVERRUN FLAG
0139 5390    389      MOV     A, @R0
013B C60F    390      ANL     A, #(OVRUN OR ERRFLG)
013B C60F    391      JZ      TILOP
013D 1F      392 ;2      INCREMENT ERROR COUNT
013D 1F      393      INC     ERRCNT
013E 2402    394 ;1      UNTIL FOREVER
013E 2402    395      JMP     TLOP
013E 2402    396 ;EOF
013E 2402    397      END

```

USER SYMBOLS

ATEMP 0007	BYFNFL 0004	DRDYFL 0008	ERRCNT 0007	ERRFLG 0010	FLGBYT 0006	MARK 0080	MDATA 0021
MFLGBY 001E	MXMTBY 0023	MOKDAT 0020	MSAMCT 001D	MTCKCT 001C	MXMTBY 0022	OMARK 0015	OSPACE 0011
OVRUN 0080	PATT 0006	RCV000 0017	KCV010 0024	RCV020 0033	RCV030 0038	RCV040 0042	KCV045 0054
RCV050 0059	RCV060 005F	RCV070 0061	RCVFLG 0001	REG0 0000	RETURN 0037	SAMCTR 0005	SPACE FF7F
STPBT5 0000	SVNFLG 0002	TCKCTR 0004	TEST 0100	TESTA 0122	TESTB 0136	TILOP 010F	TIMECNT FF7E
TISR 0007	TLOP 0102	TREC 0124	TRECE 0138	TRNGFL 0040	TRROFL 0020	UART 000A	XMI1 0064
XMT010 006E	XMT020 007B	XMT040 0086					

ASSEMBLY COMPLETE. NO ERRORS

Figure 4 (continued)

All mnemonics copyrighted © Intel Corporation 1979.

MULTIPLY ALGORITHMS

Most microcomputer programmers have at one time or another implemented a multiply routine as part of a larger program. The usual procedure is to find an algorithm that works and modify it to work on the machine being used. There is nothing wrong with this approach. If engineers felt that they had to reinvent the wheel every time a new design is undertaken, that's probably what most of us would be doing—designing wheels. If the efficiency of the multiply algorithm, either in terms

of code size or execution time is important, however, it is necessary to be reasonably familiar with the multiplication process so that appropriate optimizations for the machine being used can be made.

To understand how multiplication operates in the binary number system, consider the multiplication of two four bit operands A and B. The "ones and zeros" in A and B represent the coefficients of two polynomials. The operation $A \times B$ can be represented as the following multiplication of polynomials:

$$\begin{array}{r}
 \begin{array}{ccccccc}
 & A^3 \cdot 2^3 & + & A^2 \cdot 2^2 & + & A^1 \cdot 2^1 & + & A^0 \cdot 2^0 \\
 \times & B^3 \cdot 2^3 & + & B^2 \cdot 2^2 & + & B^1 \cdot 2^1 & + & B^0 \cdot 2^0 \\
 \hline
 & & & & & & & & & B^0 A^3 \cdot 2^3 & + & B^0 A^2 \cdot 2^2 & + & B^0 A^1 \cdot 2^1 & + & B^0 A^0 \cdot 2^0 \\
 & & & & & & & & + & B^1 A^3 \cdot 2^4 & + & B^1 A^2 \cdot 2^3 & + & B^1 A^1 \cdot 2^2 & + & B^1 A^0 \cdot 2^1 \\
 & & & & & & & & + & B^2 A^3 \cdot 2^5 & + & B^2 A^2 \cdot 2^4 & + & B^2 A^1 \cdot 2^3 & + & B^2 A^0 \cdot 2^2 \\
 & & & & & & & & + & B^3 A^3 \cdot 2^6 & + & B^3 A^2 \cdot 2^5 & + & B^3 A^1 \cdot 2^4 & + & B^3 A^0 \cdot 2^3
 \end{array}
 \end{array}$$

The sum of all these terms represents the product of A and B. The simplest multiply algorithm factors the above terms as follows:

$$A * B = B_0(A) * 2^0 + B_1(A) * 2^1 + B_2(A) * 2^2 + B_3(A) * 2^3$$

Since the coefficients of B (i.e., B₀, B₁, B₂, and B₃) can only take on the binary values of 1 or 0, the sum of the products can be formed by a series of simple adds and multiplications by two. The simplest implementation of this would be:

```
MULTIPLY:
  PRODUCT = 0
  IF B0 = 1 THEN PRODUCT = PRODUCT + A
  IF B1 = 1 THEN PRODUCT = PRODUCT + 2 * A
  IF B2 = 1 THEN PRODUCT = PRODUCT + 4 * A
  IF B3 = 1 THEN PRODUCT = PRODUCT + 8 * A
END MULTIPLY
```

In order to conserve memory, the above straight line code is normally converted to the following loop:

```
MULTIPLY:
  PRODUCT = 0
  COUNT = 4
  REPEAT
    IF B[0] = 1 THEN PRODUCT = PRODUCT + A
    A = 2 * A
    B = B / 2
    COUNT = COUNT - 1
  UNTIL COUNT = 0
END MULTIPLY
```

The repeated multiplication of A by two (which can be performed by a simple left shift) forms the terms 2 * A, 4 * A, and 8 * A. The variable B is divided by two (performed by a simple right shift) so that the least significant bit can always be used to determine whether the addition should be executed during each pass through the loop. It is from these shifting and addition opera-

tions that the "shift and add" algorithm takes its common name.

The "shift and add" algorithm shown above has two areas where efficiency will be lost if implemented in the manner shown. The first problem is that the addition to the partial product is double precision relative to the two operands. The other problem, which is also related to double precision operations, is that the A operand is double precision and that it must be left shifted and then the B operand must be right shifted. An examination of the "longhand" polynomial multiplication will reveal that, although the partial product is indeed double precision, each addition performed is only single precision. It would be desirable to be able to shift the partial product as it is formed so that only single precision additions are performed. This would be especially true if the partial product could be shifted into the "B" operand since one bit of the partial product is formed during each pass through the loop and (happily) one bit of the "B" operand is vacated. To do this, however, it is necessary to modify the algorithm so that both of the shifts that occur are of the same type.

To see how this can be done one can take the basic multiplication equation already presented:

$$A * B = B_0(A * 2^0) + B_1(A * 2^1) + B_2(A * 2^2) + B_3(A * 2^3)$$

and factoring 2⁴ from the right side:

$$A * B = 2^4 [B_0(A * 2^{-4}) + B_1(A * 2^{-3}) + B_2(A * 2^{-2}) + B_3(A * 2^{-1})]$$

This operation has resulted in a term (within the brackets) which can be formed by right shifts and adds and then multiplied by 2⁴ to get the final result. The resulting algorithm, expanded to form an eight by eight multiplication, is shown in figure 5. Note that although the result is a full sixteen bits, the algorithm only performs eight bit additions and that only a single sixteen bit shift operation is involved. This has the effect of reducing both the code space and the execution time for the routine.

ISIS-II MCS-48/UPI-41 MACRO ASSEMBLER, V2.0

LOC	OBJ	SEQ	SOURCE STATEMENT
		1	\$MACROFILE
		2	\$INCLUDE(<F1:MPV8.HED>)
		3	*****
		4	;
		5	;
		6	;
		7	*****
		8	;
		9	;
		10	;
		11	;
		12	;
		13	;

MPV8X8

THIS UTILITY PROVIDES AN 8 BY 8 UNSIGNED MULTIPLY AT ENTRY:

A = LOWER EIGHT BITS OF DESTINATION OPERAND

XA= DON'T CARE

R1= POINTER TO SOURCE OPERAND (MULTIPLIER) IN INTERNAL MEMORY

Figure 5

LOC	OBJ	SEQ	SOURCE STATEMENT
		= 14 ;*	
		= 15 ;*	AT EXIT:
		= 16 ;*	A = LOWER EIGHT BITS OF RESULT
		= 17 ;*	XA = UPPER EIGHT BITS OF RESULT
		= 18 ;*	C = SET IF OVERFLOW ELSE CLEARED
		= 19 ;*	
		= 20 ;*****	
		21 ;	
		22 ;	
		23 \$INCLUDE(:F1:MPY8.PDL)	
		= 24 ;1 MPY8X8:	
		= 25 ;1 MULTIPLICAND[15-8]:=0	
		= 26 ;1 COUNT:=8	
		= 27 ;1 REPEAT	
		= 28 ;2 IF MULTIPLICAND[0]=0 THEN BEGIN	
		= 29 ;3 MULTIPLICAND:=MULTIPLICAND/2	
		= 30 ;2 ELSE	
		= 31 ;3 MULTIPLICAND[15-8]:=MULTIPLICAND[15-8]+MULTIPLIER	
		= 32 ;3 MULTIPLICAND:=MULTIPLICAND/2	
		= 33 ;2 ENDIF	
		= 34 ;2 COUNT:=COUNT-1	
		= 35 ;1 UNTIL COUNT=0	
		= 36 ;1 END MPY8X8	
		37 ;	
		38 ; EQUATES	
		39 ; =====	
		40 ;	
0002		41 XA EQU R2	
0003		42 COUNT EQU R3	
0004		43 ICNT EQU R4	
		44 ;	
0003		45 DIGPR EQU 3	
		46 ;	
		47 \$EJECT	
		48 \$INCLUDE(:F1:MPY8)	
		= 49 ;1 MPY8X8:	
		= 50 MPY8X8:	
		= 51 ;1 MULTIPLICAND[15-8]:=0	
0000 BA00		= 52 MOV XA,#00	
		= 53 ;1 COUNT:=8	
0002 BB08		= 54 MOV COUNT,#8	
		= 55 ;1 REPEAT	
		= 56 MPY8LP:	
		= 57 ;2 IF MULTIPLICAND[0]=0 THEN BEGIN	
0004 120E		= 58 JBO MPY8A	
		= 59 ;3 MULTIPLICAND:=MULTIPLICAND/2	
0006 2A		= 60 XCH A,XA	
0007 97		= 61 CLR C	
0008 67		= 62 RRC A	
0009 2A		= 63 XCH A,XA	
000A 67		= 64 RRC A	
000B EB04		= 65 DJNZ COUNT,MPY8LP	
000D 83		= 66 RET	
		= 67 ;2 ELSE	

Figure 5 (continued)

LOC	OBJ	SEQ	SOURCE STATEMENT
		= 68	MPY8A:
		= 69 ; 3	MULTIPlicAND[15-8]:=MULTIPlicAND[15-8]+MULTIPLIER
000E 2A		= 70	XCH A, XA
000F 61		= 71	ADD A, @R1
0010 67		= 72	RRC A
0011 2A		= 73	XCH A, XA
0012 67		= 74	RRC A
0013 E004		= 75	DJNZ COUNT, MPY8LP
0015 83		= 76	RET
		= 77 ; 3	MULTIPlicAND:=MULTIPlicAND/2
		= 78 ; 2	ENDIF
		= 79 ; 2	COUNT:=COUNT-1
		= 80 ; 1 UNTIL COUNT=0	
		= 81 ; 1 END MPY8X3	
		= 82	END

USER SYMBOLS

COUNT	0003	DIGPR	0003	ICNT	0004	MPY8A	000E	MPY8LP	0004	MPY8X3	0000	XA	0002
-------	------	-------	------	------	------	-------	------	--------	------	--------	------	----	------

ASSEMBLY COMPLETE. NO ERRORS

All mnemonics copyrighted © Intel Corporation 1979.

DIVIDE ALGORITHMS

In order to understand binary division a four bit operation will again be used as an example. The following algorithm will perform a four by four division:

```

DIVIDE:
  IF 16*DIVISOR>= DIVIDEND THEN
    SET OVERFLOW ERROR FLAG
  ELSE
    IF 8*DIVISOR>= DIVIDEND THEN
      QUOTIENT[3]:= 1
      DIVIDEND:= DIVIDEND - 8*DIVISOR
    ELSE
      QUOTIENT[3]:= 0
    ENDIF
    IF 4*DIVISOR>= DIVIDEND THEN
      QUOTIENT[2]:= 1
      DIVIDEND:= DIVIDEND - 4*DIVISOR
    ELSE
      QUOTIENT[2]:= 0
    ENDIF
    IF 2*DIVISOR>= DIVIDEND THEN
      QUOTIENT[1]:= 1
      DIVIDEND:= DIVIDEND - 2*DIVISOR
    ELSE
      QUOTIENT[1]:= 0
    ENDIF
    IF 1*DIVISOR>= DIVIDEND THEN
      QUOTIENT[0]:= 1
      DIVIDEND:= DIVIDEND - 1*DIVISOR
    ELSE
      QUOTIENT[0]:= 0
    ENDIF
  ENDIF
END DIVIDE

```

The algorithm is easy to understand. The first test asks if the division will fit into the dividend sixteen times. If it will, the quotient cannot be expressed in only four bits so an overflow error flag is set and the divide algorithm ends. The algorithm then proceeds to determine if eight times the divisor fits, four times, etc. After each test it either sets or clears the appropriate quotient bit and modifies the dividend. To see this algorithm in action, consider the division of 15 by 5:

00001111	(15)
- 01010000	(16*5)
<hr/>	
00001111	(15)
- 00101000	(8*5)
<hr/>	
00001111	(15)
- 00010100	(4*5)
<hr/>	
00001111	(15)
- 00001010	(2*5)
<hr/>	
00000101	Fits—Q[1]= 1
<hr/>	
00000101	(15-2*5)
- 00000101	(1*5)
<hr/>	
00000000	Fits—Q[0]= 1

The result is Q=0011 which is the binary equivalent of 3—the correct answer. Clearly this algorithm can (and has been) converted to a loop and used to perform divisions. An examination of the procedure, however, will show that it has the same problems as the original multiply algorithm.

The first problem is that double precision operations are involved with both the comparison of the division with the dividend and the conditional subtraction. The second problem is that as the quotient bits are derived they must be shifted into a register. In order to reduce the register requirements, it would be desirable to shift them into the divisor register as they are generated since the divisor register gets shifted anyway. Unfortunately the quotient bits are derived most significant bits first so doing this will form a mirror image of the quotient—not very useful.

Both of these problems can be solved by observing that the algorithm presented for divide will still work if both sides of all the “equations” involving the dividend are divided by sixteen. The looping algorithm then would proceed as follows:

```
DIVIDE:
QUOTIENT:= 0
COUNT:= 4
DIVIDEND:= DIVIDEND/16
IF DIVISOR>= DIVIDEND THEN
    OVERFLOW FLAG:= 1
ELSE
    REPEAT
        DIVIDEND:= DIVIDEND*2
        QUOTIENT:= QUOTIENT*2
        IF DIVISOR>= DIVIDEND THEN
            QUOTIENT:= QUOTIENT + 1/*SET QUOTIENT[0]*/
            DIVIDEND:= DIVIDEND - DIVISOR
        ENDIF
        COUNT:= COUNT - 1
    UNTIL COUNT= 0
    ENDIF
END DIVIDE
```

When this algorithm is implemented on a computer which does not have a direct compare instruction the comparison is done by subtraction and the inner loop of the algorithm is modified as follows:

```
REPEAT
    DIVIDEND:= DIVIDEND*2
    QUOTIENT:= QUOTIENT*2
    DIVIDEND:= DIVIDEND - DIVISOR
    IF BORROW= 0 THEN
        QUOTIENT:= QUOTIENT + 1
    ELSE
        DIVIDEND:= DIVIDEND + DIVISOR
    ENDIF
    COUNT:= COUNT - 1
UNTIL COUNT= 0
*
```

An implementation of this algorithm using the 8049 instruction set is shown in figure 6. This routine does an unsigned divide of a 16 bit quantity by an eight bit quantity. Since the multiply algorithm of figure 5 generates a 16 bit result from the multiplication of two eight bit operands, these two routines complement each other and can be used as part of more complex computations.

IS15-II MCS-48/UPI-41 MACRO ASSEMBLER, V2.0

LOC	OBJ	SEQ	SOURCE STATEMENT
		1	\$MACROFILE
		2	\$INCLUDE(:F1:DIV16. HED)
		3	*****
		4	;
		5	;
		6	;
		7	*****
		8	;
		9	;
		10	;
		11	;
		12	;
		13	;
		14	;
		15	;
		16	;
		17	;

Figure 6

LOC	OBJ	SEQ	SOURCE STATEMENT	DISASSEMBLY	HEX	DEC
		= 18 ;*	C = SET IF OVERFLOW ELSE CLEARED			*
		= 19 ;*				*
		= 20 ;*****				
		21 ;				
		22 ;				
		23 #INCLUDE(:F1:DIV16.PDL)				
		= 24 ;1 DIV16:				
		= 25 ;1 COUNT:=8				
		= 26 ;1 DIVIDEND[15-8]:=DIVIDEND[15-8]-DIVISOR				
		= 27 ;1 IF BORROW=0 THEN /* IT FITS*/				
		= 28 ;2 SET OVERFLOW FLAG				
		= 29 ;1 ELSE				
		= 30 ;2 RESTORE DIVIDEND				
		= 31 ;2 REPEAT				
		= 32 ;3 DIVIDEND:=DIVIDEND*2				
		= 33 ;3 QUOTIENT:=QUOTIENT*2				
		= 34 ;3 DIVIDEND[15-8]:=DIVIDEND[15-8]-DIVISOR				
		= 35 ;3 IF BORROW=1 THEN				
		= 36 ;4 RESTORE DIVIDEND				
		= 37 ;3 ELSE				
		= 38 ;4 QUOTIENT[0]:=1				
		= 39 ;3 ENDIF				
		= 40 ;3 COUNT:=COUNT-1				
		= 41 ;2 UNTIL COUNT=0				
		= 42 ;2 CLEAR OVERFLOW FLAG				
		= 43 ;1 ENDIF				
		= 44 ;1 ENDDIVIDE				
		45 ;				
		46 ; EQUATES				
		47 ; =====				
		48 ;				
0002		49 XA EQU R2				
0003		50 COUNT EQU R3				
		51 ;				
		52 \$EJECT				
		53 #INCLUDE(:F1:DIV16)				
		= 54 ;1 DIV16:				
0000 2A		= 55 DIV16: XCH A,XA ; ROUTINE WORKS MOSTLY WITH BITS 15-8				
		= 56 ;1 COUNT:=8				
0001 BB08		= 57 MOV COUNT,#8				
		= 58 ;1 DIVIDEND[15-8]:=DIVIDEND[15-8]-DIVISOR				
0003 37		= 59 CPL A				
0004 61		= 60 ADD A,@R1				
0005 37		= 61 CPL A				
		= 62 ;1 IF BORROW=0 THEN /* IT FITS*/				
0006 F60B		= 63 JC DIVIA				
		= 64 ;2 SET OVERFLOW FLAG				
0008 A7		= 65 CPL C				
0009 0424		= 66 JMP DIVIB				
		= 67 ;1 ELSE				
		= 68 DIVIA:				
		= 69 ;2 RESTORE DIVIDEND				
000B 61		= 70 ADD A,@R1				
		= 71 ;2 REPEAT				
		= 72 DIVILP:				
		= 73 ;3 DIVIDEND:=DIVIDEND*2				

Figure 6 (continued)

```

000C 97      = 75      CLR      C
000D 2A      = 76      XCH      A,XA
000E F7      = 77      RLC      A
000F 2A      = 78      XCH      A,XA
0010 F7      = 79      RLC      A
0011 E618    = 80      JNC      DIVIE
0012 27      = 81      CPL      A
0014 61      = 82      ADD      A,@R1
0015 37      = 83      CPL      A
0016 0420    = 84      JMP      DIVIC
              = 85 :3    DIVIDEND[15-8]:=DIVIDEND[15-8]-DIVISOR
0018 37      = 86 DIVIE: CPL      A
0019 61      = 87      ADD      A,@R1
001A 37      = 88      CPL      A
              = 89 :3    IF BORROW=1 THEN
001B E620    = 90      JNC      DIVIC
              = 91 :4    RESTORE DIVIDEND
001D 61      = 92      ADD      A,@R1
001E 0421    = 93      JMP      DIVID
              = 94 :3    ELSE
              = 95 DIVIC:
              = 96 :4    QUOTIENT[0]:=1
0020 1A      = 97      INC      XA
              = 98 :3    ENDIF
              = 99 :3    COUNT:=COUNT-1
              = 100 :2   UNTIL COUNT=0
0021 E80C    = 101 DIVID: DJNZ    COUNT,DIVILP
              = 102 :2   CLEAR OVERFLOW FLAG
0023 97      = 103     CLR      C
              = 104 :1   ENDIF
              = 105 :1   ENDDIVIDE
0024 2A      = 106 DIVIB: XCH      A,XA
0025 83      = 107     RET
              108 END

```

USER SYMBOLS

```

COUNT 0003  DIV16 0000  DIV1A 000B  DIV1B 0024  DIVIC 0020  DIVID 0021  DIVIE 0018  DIVILP 000C
XA       0002

```

ASSEMBLY COMPLETE, NO ERRORS

Figure 6 (continued)

All mnemonics copyrighted © Intel Corporation 1979.

BINARY AND BCD CONVERSIONS

The conversion of a binary value to a BCD (binary coded decimal) number can be done with a very straightforward algorithm:

```

CONVERT_TO_BCD:
BCDACCUM:=0
COUNT:=PRECISION
REPEAT
    BIN:=BIN * 2
    BCD:=BCD * 2 + CARRY
    COUNT:=COUNT - 1
UNTIL COUNT=0
END CONVERT_TO_BCD

```

The variable **BCDACCUM** is a BCD string used to accumulate the result; the variable **BIN** is the binary number to be converted. **PRECISION** is a constant which gives the length, in binary bits of **BIN**. To see how this works, assume that **BIN** is a sixteen bit value with the most significant bit set. On the first pass through the loop the multiplication of **BIN** will result in a carry and this carry will be added to BCD. On the remaining passes through the loop BCD will be multiplied by two 15 times. The initial carry into BCD will be multiplied by 2^{15} or 32678, which is the "value" of the most significant bit of **BIN**. The process repeats with each bit of **BIN** being introduced to **BCDACCUM** and then being scaled up on successive passes through the loop. Figure 7 shows the implementation of this algorithm for the 8049.

LOC	OBJ	SEQ	SOURCE STATEMENT
		1	*\$MACROFILE
		2	*\$INCLUDE(:F1:CONBCD.HED)
		3	*****
		4	;
		5	CONBCD
		6	;
		7	*****
		8	;
		9	THIS UTILITY CONVERTS A 16 BIT BINARY VALUE TO BCD
		10	AT ENTRY:
		11	A = LOWER EIGHT BITS OF BINARY VALUE
		12	XA= UPPER EIGHT BITS OF BINARY VALUE
		13	R0= POINTER TO A PACKED BCD STRING
		14	;
		15	AT EXIT:
		16	A = UNDEFINED
		17	XA= UNDEFINED
		18	C = SET IF OVERFLOW ELSE CLEARED
		19	;
		20	*****
		21	;
		22	;
		23	*\$INCLUDE(:F1:CONBCD.PDL)
		24	1 CONVERT_TO_BCD
		25	1 BCDACC:=0
		26	1 COUNT:=16
		27	1 REPEAT
		28	2 BIN:=BIN*2
		29	2 BCD:=BCD*2+CARRY
		30	2 IF CARRY FROM BCDACC GOTO ERROR EXIT
		31	2 COUNT:=COUNT-1
		32	1 UNTIL COUNT=0
		33	1 END CONVERT_TO_BCD
		34	;
		35	EQUATES
		36	*****
		37	;
0002		38	XA EQU R2
0003		39	COUNT EQU R3
0004		40	ICNT EQU R4
		41	;
0003		42	DIGPR EQU 3
		43	;
		44	\$EJECT
		45	*\$INCLUDE(:F1:CONBCD)
		46	;
0005		47	TEMP1 SET R5
		48	;
		49	1 CONVERT_TO_BCD
		50	CNBCD:
		51	1 BCDACC:=0
0000 28		52	XCH A,R0

Figure 7

LOC	OBJ	SEQ	SOURCE STATEMENT						
0001	A0	= 53	MOV R1, A						
0002	28	= 54	XCH A, R0						
0003	BC03	= 55	MOV ICNT, #DIGPR						
0005	B100	= 56	BCDCOR: MOV @R1, #00						
0007	19	= 57	INC R1						
0008	EC05	= 58	DJNZ ICNT, BCDCOR						
		= 59	:1 COUNT:=16						
000A	BB10	= 60	MOV COUNT, #16						
		= 61	:1 REPEAT						
		= 62	BCDCOB:						
		= 63	:2 BIN:=BIN*2						
000C	97	= 64	CLR C						
000D	F7	= 65	RLC A						
000E	2A	= 66	XCH A, XA						
000F	F7	= 67	RLC A						
0010	2A	= 68	XCH A, XA						
		= 69	:2 BCD:=BCD*2+CARRY						
0011	28	= 70	XCH A, R0						
0012	A9	= 71	MOV R1, A						
0013	28	= 72	XCH A, R0						
0014	BC03	= 73	MOV ICNT, #DIGPR						
0016	AD	= 74	MOV TEMP1, A						
0017	F1	= 75	BCDOC: MOV A, @R1						
0018	71	= 76	ADDC A, @R1						
0019	57	= 77	DA A						
001A	A1	= 78	MOV @R1, A						
001B	19	= 79	INC R1						
001C	EC17	= 80	DJNZ ICNT, BCDOC						
001E	FD	= 81	MOV A, TEMP1						
		= 82	:2 IF CARRY FROM BCDACC GOTO ERROR EXIT						
001F	F624	= 83	JC BCDCOB						
		= 84	:2 COUNT:=COUNT-1						
		= 85	:1 UNTIL COUNT=0						
0021	E80C	= 86	DJNZ COUNT, BCDCOB						
0023	97	= 87	CLR C ; CLEAR CARRY TO INDICATE NORMAL TERMINATION						
		= 88	:1 END CONVERT_TO_BCD						
0024	83	= 89	BCDCOD: RET						
		= 90	END						
USER SYMBOLS									
BCDCOR	0005	BCDCOB	000C	BCDCOD	0024	BCDOC	0017	CNBCD	0000
TEMP1	0005	XA	0002						
ASSEMBLY COMPLETE, NO ERRORS									

Figure 7 (continued)

The conversion of a BCD value to binary is essentially the same process as converting a binary value to BCD.

```

CONVERT_TO_BINARY
  BIN:=0
  COUNT:= DIGNO
  REPEAT
    BCDACCUM:= BCDACCUM * 10
    BIN:= 10 * BIN + CARRY DIGIT
    COUNT:= COUNT - 1
  UNTIL COUNT = 0
END CONVERT_TO_BINARY

```

The only complexity is the two multiplications by ten. The BCDACCUM can be multiplied by ten by shifting it left four places (one digit). The variable BIN could be multiplied using the multiply algorithm already discussed, but it is usually more efficient to do this by mak-

ing the following substitution:

$$\text{BIN} = 10 * \text{BIN} = (2) * (5) * (\text{BIN}) = 2 * (2 * 2 + 1) * \text{BIN}$$

This implies that the value $10 * \text{BIN}$ can be generated by saving the value of BIN and then shifting BIN two places left. After this the original value of BIN can be added to the new value of BIN (forming $5 * \text{BIN}$) and then BIN can be multiplied by two. It is often possible to implement the multiplication of a value by a constant by using such techniques. Figure 8 shows an 8049 routine which converts BCD values to binary. This routine differs slightly from the algorithm above in that the BCD digits are read, and converted to binary, two digits at a time. Protection has also been added to detect BCD operands which, if converted, would yield binary values beyond the range of the result.

ISIS-II MCS-48/UPI-41 MACRO ASSEMBLER, V2.0

LOC	OBJ	SEQ	SOURCE STATEMENT
		1	\$MACROFILE
		2	\$INCLUDE(:F1:CONBIN.HED)
		3	*****
		4	;
		5	;
		6	;
		7	*****
		8	;
		9	;
		10	THIS UTILITY CONVERTS A 6 DIGIT BCD VALUE TO BINARY
		11	AT ENTRY:
		12	RO= POINTER TO A PACKED BCD STRING
		13	AT EXIT:
		14	A = LOWER EIGHT BITS OF THE BINARY RESULT
		15	XR= UPPER EIGHT BITS OF THE BINARY RESULT
		16	C = SET IF OVERFLOW ELSE CLEARED
		17	*****
		18	*****
		19	;
		20	;
		21	\$INCLUDE(:F1:CONBIN.PDL)
		22	;
		23	;
		24	1 CONVERT_TO_BINARY
		25	1 POINTER0:=POINTER0+DIGITPAIR-1
		26	1 COUNT:=DIGITPAIR
		27	1 BIN:=0
		28	1 REPEAT
		29	2 BIN:=BIN*10
		30	2 BIN:=BIN+MEM(R0)[7-4]
		31	2 BIN:=BIN*10
		32	2 BIN:=BIN+MEM(R0)[3-0]

LOC	OBJ	SEQ	SOURCE STATEMENT
		= 33 ; 2	POINTER0:=POINTER0-1
		= 34 ; 2	COUNT:=COUNT-1
		= 35 ; 1	UNTIL COUNT=0
		= 36 ; 1	END CONVERT_TO_BINARY
		= 37 ;	
		= 38 ;	EQUATES
		= 39 ;	=====
		= 40 ;	
0002		41 XA	EQU R2
0003		42 COUNT	EQU R3
0004		43 ICNT	EQU R4
		= 44 ;	
0003		45 DIGPR	EQU 3
		= 46 ;	
		= 47	\$EJECT
		= 48	\$INCLUDE(:F1:CONBIN)
		= 49 ;	
0005		= 50 TEMP1	SET R5
0006		= 51 TEMP2	SET R6
		= 52 ;	
		= 53 ; 1	CONVERT_TO_BINARY
		= 54	CONBIN:
		= 55 ; 1	POINTER0:=POINTER0+DIGITPAIR-1
0000 F8		= 56	MOV A, R0
0001 0302		= 57	ADD A, #DIGPR-1
0003 A8		= 58	MOV R0, A
		= 59 ; 1	COUNT:=DIGITPAIR
0004 BB03		= 60	MOV COUNT, #DIGPR
		= 61 ; 1	BIN:=0
0006 27		= 62	CLR A
0007 AA		= 63	MOV XA, A
		= 64 ; 1	REPEAT
		= 65	CONBLP:
		= 66 ; 2	BIN:=BIN*10
0008 142B		= 67	CALL CONB10
000A F62A		= 68	JC CONBER
		= 69 ; 2	BIN:=BIN+MEM(R0)[7-4]
000C AD		= 70	MOV TEMP1, A
000D F0		= 71	MOV A, @R0
000E 47		= 72	SWAP A
000F 530F		= 73	ANL A, #0FH
0011 6D		= 74	ADD A, TEMP1
0012 2A		= 75	XCH A, XA
0013 1300		= 76	ADDC A, #00
0015 2A		= 77	XCH A, XA
0016 F62A		= 78	JC CONBER
		= 79 ; 2	BIN:=BIN*10
0018 142B		= 80	CALL CONB10
001A F62A		= 81	JC CONBER
		= 82 ; 2	BIN:=BIN+MEM(R0)[3-0]
001C AD		= 83	MOV TEMP1, A
001D F0		= 84	MOV A, @R0
001E 530F		= 85	ANL A, #0FH
0020 6D		= 86	ADD A, TEMP1
0021 2A		= 87	XCH A, XA

LOC	OBJ	SEQ	SOURCE STATEMENT
0022	1300	= 88	ADDC A, #00
0024	2A	= 89	XCH A, XA
0025	F62A	= 90	JC CONBER
		= 91 ; 2	POINTER0:=POINTER0-1
0027	C8	= 92	DEC R0
		= 93 ; 2	COUNT:=COUNT-1
		= 94 ; 1	UNTIL COUNT=0
0028	EB08	= 95	DJNZ COUNT, CONBLP
		= 96 ; 1	END CONVERT_TO_BINARY
002A	83	= 97	CONBER: RET
		= 98	#EJECT
		= 99 ;	
		= 100 ;	
		= 101 ;	UTILITY TO MULTIPLY BIN BY 10
		= 102 ;	CARRY WILL BE SET IF OVERFLOW OCCURS
		= 103 ;	
002B	AD	= 104	CONB10: MOV TEMP1, A ; SAVE A
002C	2A	= 105	XCH A, XA ; SAVE XA
002D	AE	= 106	MOV TEMP2, A
002E	2A	= 107	XCH A, XA
		= 108 ;	
002F	97	= 109	CLR C
0030	F7	= 110	RLC A ; BIN:=BIN*2
0031	2A	= 111	XCH A, XA
0032	F7	= 112	RLC A
0033	2A	= 113	XCH A, XA
0034	F646	= 114	JC CONB1E ; ERROR ON OVERFLOW
		= 115 ;	
0036	F7	= 116	RLC A ; BIN:=BIN*4
0037	2A	= 117	XCH A, XA
0038	F7	= 118	RLC A
0039	2A	= 119	XCH A, XA
003A	F646	= 120	JC CONB1E ; ERROR ON OVERFLOW
		= 121 ;	
003C	6D	= 122	ADD A, TEMP1 ; BIN:=BIN*5
003D	2A	= 123	XCH A, XA
003E	7E	= 124	ADDC A, TEMP2
003F	2A	= 125	XCH A, XA
0040	F646	= 126	JC CONB1E ; ERROR ON OVERFLOW
		= 127 ;	
0042	F7	= 128	RLC A ; BIN:=BIN*10
0043	2A	= 129	XCH A, XA
0044	F7	= 130	RLC A
0045	2A	= 131	XCH A, XA
		= 132 ;	
0046	83	= 133	CONB1E: RET
		= 134	
		= 135 ;	
		= 136	END

USER SYMBOLS

CONB10 002B	CONB1E 0046	CONBER 002A	CONBIN 0000	CONBLP 0008	COUNT 0003	DIGPR 0003	1CNT 0004
TEMP1 0005	TEMP2 0006	XA 0002					

ASSEMBLY COMPLETE, NO ERRORS

CONCLUSION

The design goals of the full duplex serial communication software were realized. If transmission and reception are occurring concurrently, only 45 percent of the real time available to the 8048 will be consumed by the serial link. This implies that an 8048 running full duplex serial I/O will still outperform earlier members of the family running without the serial I/O requirement. It is also possible to run this program in an 8048 or 8748 at 1200 baud with the same 45 percent CPU utilization.

The execution times for the other routines that have been discussed have been summarized in Table I. All of these routines were written to maintain maximum efficiency rather than minimum code size or execution time. The resulting execution times and code size are therefore what the user can expect to see in a real application. The results that were obtained clearly show the advantage and speed of the 8048. The equivalent 8085 code for the 8048 is also shown. It is clear that the 8048 requires a substantial performance advantage over the 8085. Considering in most applications that the 8048 is

real time available to the 8049 will be consumed by the serial link. This implies that an 8049 running full duplex serial I/O will still outperform earlier members of the family running without the serial I/O requirement. It is also possible to run this program in an 8048 or 8748 at 1200 baud with the same 42 percent CPU utilization.

The execution times for the other routines that have been discussed have been summarized in Table 1. All of these routines were written to maintain maximum useability rather than minimum code size or execution time. The resulting execution times and code size are therefore what the user can expect to see in a real application. The results that were obtained clearly show the efficiency and speed of the 8049. The equivalent times for the 8048 are also shown. It is clear that the 8049 represents a substantial performance advantage over the 8048. Considering, in most applications, that the 8048 is

have required too much computer power for a single chip approach.

EXECUTION TIME (MICROSECONDS)

	BYTES	8049	8048
MPY8	21	109	200
DIV 16	37	183 MIN 204 MAX	335 MIN 375 MAX
CONBCD	36	733	1348
CONBIN	70	388	713

Table 1. Program Performance

Contents	
I. Purpose and Scope	5-2
II. The HSE-48™ Development Tool	5-3
III. General Hardware Overview	5-3
IV. Interprocessor Communication	5-5
V. Command Description	5-6
VI. System Limitations	5-9
VII. Hardware Configurations	5-10
Appendix A. Schematic Diagrams	5-12
Appendix B. Memory Mapping	5-16
Appendix C. Command Summary	5-14
Appendix D. Error Messages	5-14

A High-Speed Emulator for Intel MCS-48™ Microcomputers

August 1979

A High-Speed Emulator for Intel MCS-48™ Microcomputers

Applications Staff
Microcontroller Operation

A High-Speed Emulator for Intel MCS-48™ Microcomputers

Contents

I. Purpose and Scope	5-3
II. The HSE-49™ Development Tool	5-3
III. General Hardware Overview	5-3
IV. Interprocessor Communication	5-5
V. Command Description	5-6
VI. System Limitations	5-9
VII. Hardware Configurations	5-10
Appendix A. Schematic Diagrams	5-12
Appendix B. Monitor Listings	5-16
Appendix C. Command Summary	5-104
Appendix D. Error Messages	5-104

I. PURPOSE AND SCOPE

This Application Note presents a description of the design and operation of a high-speed emulator for the Intel® MCS-48™ family of single chip microcomputers. The HSE-49™ emulator provides a simple and inexpensive means for executing and debugging 8049 programs which require the full 11-MHz operating speed of the part.

Section II of this Application Note describes some of the features of this development tool and how it may be used. Section III briefly discusses the hardware used to implement these features, while Section IV describes the manner in which program execution status is made available to the operator.

A detailed description of all of the operator commands is presented in Section V of this note, along with the modifiers and options which may be specified for each command. Known restrictions and limitations of the HSE-49 system are listed and explained in Section VI. Section VII shows how the basic circuit may be modified to provide options on memory organization, I/O configurations, etc.

Full schematics of the system hardware, as well as monitor software listings, are presented in Appendices A and B, respectively. A short summary of the command syntax is presented in Appendix C. Appendix D explains the error message codes which may appear during use.

It is assumed that the reader is already familiar with the operation of the 8048 or 8049 microcomputers. Some knowledge of the 8048 architecture is needed to understand sections of the command and modifier descriptions. Most users will already have this background. Other readers are referred to the *MCS-48 Microcomputer User's Manual*, Intel publication number 9800270.

II. THE HSE-49 DEVELOPMENT TOOL

In essence, the HSE-49 emulator provides the user a means for executing an MCS-48 program located in external RAM rather than internal ROM or EPROM. This allows programs being debugged to be modified easily and quickly during the debug cycle. A user's program may be entered into system RAM either manually or via a serial link from a host computer such as an Intellec® Microcomputer Development System. Once loaded, the program can be modified using an on-board keyboard and display, and executed in real-time in a number of breakpoint modes. The internal state of the processor, including RAM, accumulator, timer/counter, and status register contents, can also be read and modified through the keyboard.

Breakpoint and debug facilities are extremely flexible. The following execution modes are provided.

- Programs may be run in full (11 MHz) real time;
- Programs may be single-stepped;
- In break mode, programs run in full real time until break occurs;

- Breaks may be triggered by either program or external data RAM accesses;
- Any number of breakpoints may be used in any combination;
- "Auto-Step" operation causes the current program counter and Accumulator contents to be printed on the display for a short time on every instruction cycle;
- "Auto-Break" provides the above display only when a break flag is encountered, with real time operation otherwise;
- While running in non-break mode, a TTL-level pulse is generated whenever a break flag is encountered. This signal may be used to trigger an oscilloscope or Logic Analyzer to assist in hardware and software debug.
- While running in any mode, the keyboard and display are "alive". Execution may be suspended or terminated by commands from the keyboard.

Intent of this Note

While the HSE-49 emulator can assist a new microcomputer user in becoming familiar with the 8048 and 8049 microcomputers, its inherent debug capabilities will also prove helpful to design engineers. The design could be used for new system development and verification or adapted for prototype production.

The main concern in designing the HSE-49 emulator was to keep the basic design simple, while maximizing the system's flexibility. The design allows the use of jumpers, hardware and software switches, etc. to allow the user to reconfigure the system according to the way he dedicates chip-select pins, I/O, etc. The emulator can be changed to fit each user's unique needs, rather than forcing the user to alter his needs to what is provided.

The primary intent of note is to provide the reader with the information needed to reconstruct and make full use of the HSE-49 emulator. Less emphasis is placed on describing how the hardware operates or how the commands are implemented. This information may be found in the schematic diagrams and software listings included in the Appendices.

III. GENERAL HARDWARE OVERVIEW

User Program Emulation

The actual emulation of the user's program is done using an 8039 microcomputer (IC29 on the schematics in Appendix A) executing a program stored in external RAM. The basic minimum configuration includes the 8039 microcomputer, an 8282 address latch (IC19), and 2K bytes of 2114 RAM to use for program development and real-time execution (ICs B1, C1, B2, and C2). Additional RAM may be added to allow the user to expand his program and data memory to 4K each. (If an 11-MHz crystal is used with the microcomputer, type 2114-3 RAMs must be used.)

keyboard and display, interpret and implement commands, drive serial interfaces, etc. In general, the master processor is used to interface the execution processor's memory spaces with the outside world and control the operation of the execution processor. In this note the two processors will be abbreviated "MP" and "EP", respectively. Figure 1 shows how the two processors interrelate with the rest of the system.

Keyboard/Display

The 33-key keyboard shown in Figure 2 includes a 16-key hexadecimal keypad and 17 special function keys for specifying commands and modifiers. Readers already

additional keys are used to generalize and augment the PROMPT-48 capabilities, as described in Section V.

The eight-character seven-segment display (DS1-DS8) is used for displaying addresses, data, and pseudo-alphanumeric messages. The display responses printed in Section V and throughout this note use a mix of upper and lower case letters to indicate what seven-segment patterns appear. An 8243 (IC9) and eight DIP packages (resistor packs, current buffers, etc.) are used for multiplexing the display and scanning the keyboard.

Breakpoint Detection

Breakpoints are specified and detected using a 2102A 1K x 8 RAM corresponding to each pair of 2114s (ICs A1

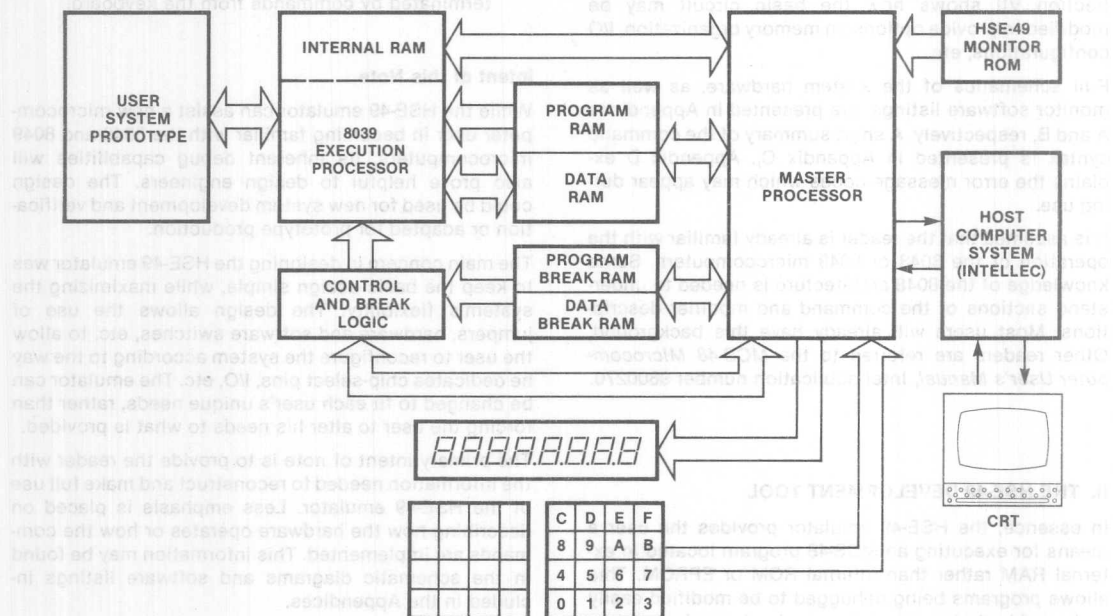


Figure 1. HSE-49™ Emulator Signal Flow Diagram

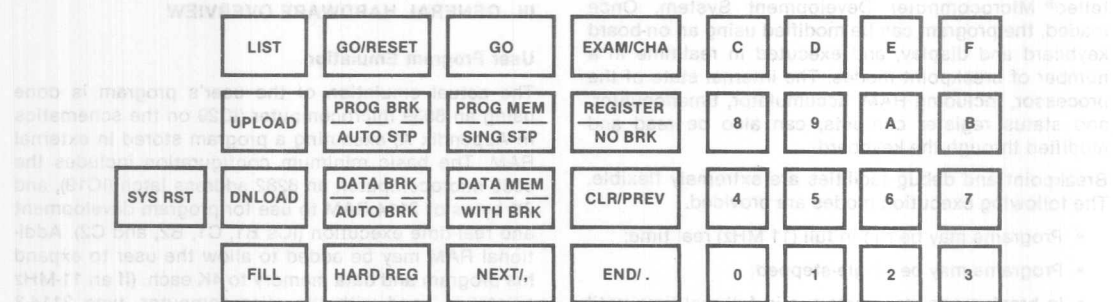


Figure 2. HSE-49™ Emulator Command Keyboard Organization

and A2). In effect, each program or data address accesses a 9-bit word. Eight bits are used normally for code or data storage. The ninth bit, accessed in parallel with the other eight, is used to indicate if a breakpoint has been set for that address. This output, when asserted, is latched (IC27 and IC36) and used to halt the execution processor via the single-step input. (In other modes, the break logic can be reconfigured to set the break requested flip-flop on any EP machine cycle or any EP "MOVX" instruction.)

Link Register

An 8212 8-bit latch (IC18) is used to communicate data and commands between the master and control processors. Under control of the MP, this register, called the "Link" register, may be logically mapped into either the program or data RAM address spaces. When this is done, the 2114s in the respective memory space are disabled and the link responds to all accesses, regardless of address. The link will be discussed in greater detail in Section IV.

Control Logic

In addition to the devices mentioned above, the minimum configuration requires about 10 additional ICs for bus arbitration, system control, and breakpoint and single-step logic. Additional parts may be optionally added for serial port interfacing, I/O reconstruction, etc.

MP Monitor

The monitor program executed by the MP includes commands for filling, reading, or writing the various memory spaces, including the execution processor's program RAM, external ("MOVX") data RAM, accumulator, PSW, PC, timer/counter, working registers, and internal RAM; to execute the user's program from arbitrary addresses in various debugging modes; and to upload or download object or data files from diskettes using an Inteltec® development system. No special software is needed for the Inteltec® other than ISIS Version 3.4 or later. The data format is compatible with the standard Intel hex file format produced by ASM-4; the baud rate may be altered from 110 baud (default state) up to 2400

baud from the on-board keybaud. Blocks of data may be transmitted to a CRT or printer and displayed in a tabular format.

IV. INTERPROCESSOR COMMUNICATION

Program Break Sequence

When the MP detects that the EP has been halted by the breakpoint hardware, or when the operator presses a key while the program is executing, the program break sequence is initiated. The low-order 23 bytes of user program memory is read into a buffer within the internal RAM of the MP. A short program for reading and transmitting internal EP status is written over the low-order program memory. (This is one of several "mini-monitors" overlaid over the user program area.) The link register is mapped logically over the user program memory, and loaded with the 8049 machine code for a "CALL" instruction to the mini-monitor program area. The EP is then allowed to fetch a single instruction from the link, i.e., the "CALL" to the mini-monitor is forced onto the EP data bus.

From this point on, the EP executes code contained in the mini-monitor. The link is logically mapped over the data RAM address space (whether or not any 2114 data RAMs are present). A block diagram of the system at this point is shown in Figure 3. The break logic is reconfigured so that any "MOVX" (RD or WR) operation executed by the EP will cause it to halt.

For example, after entering the first mini-monitor, the EP executes a "MOVX @R0,A" instruction. This writes the contents of the accumulator prior to the execution termination into the link, and causes the EP to halt. The MP may then read and retain the link contents to determine the EP accumulator value. The EP timer/counter and PSW are preserved in the same manner.

Accessing EP Internal RAM

After reading and saving EP internal status, the MP loads a different mini-monitor into the same RAM area. This monitor allows the internal RAM of the EP to be read and written by the MP by passing address and data

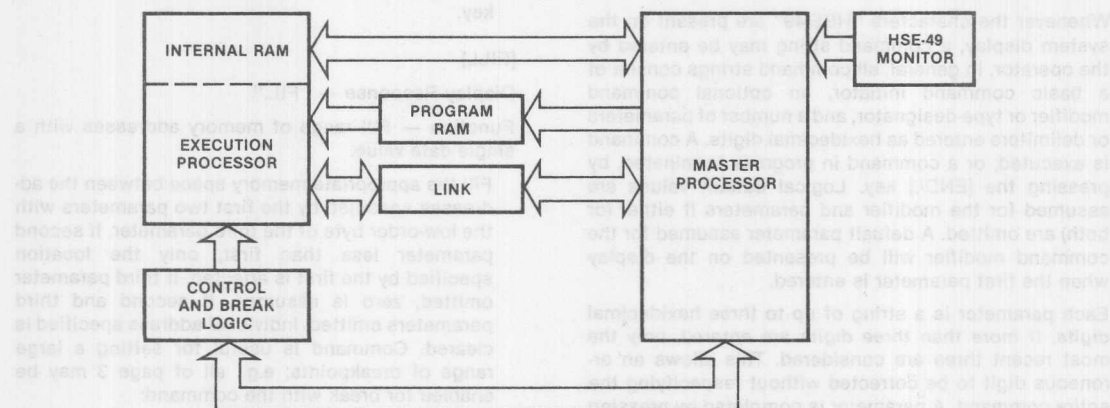


Figure 3. Communication between EP & MP

values between the two processors using the link register.

This is needed for two reasons. First, the EP program counter prior to the forced "CALL" instruction may be derived from the EP stack contents, and may be modified to cause the EP to resume execution at any desired address. Secondly, the internal RAM of the EP may then be accessed and modified in the process of executing a number of the monitor commands.

Resuming User Program Execution

In order to resume user program execution, a status-restoration mini-monitor is overlaid. This restores the EP internal status using a scheme analogous to the one in which the status was originally saved. The final step of the last mini-monitor is an "RETR" instruction, after which the EP is again halted. The low-order program memory saved earlier is rewritten into the appropriate area, the break logic is reconfigured for the desired execution mode, and the EP is released to run at full speed until the next break situation is encountered.

Note that all commands are implemented using "logical" rather than "physical" addressing. Thus the operator need not be concerned with the intricacies of the system design. For example, when any monitor command refers to low-order user program memory, the appropriate byte of storage within the MP internal RAM is accessed instead. If the location is altered, the internal RAM is modified appropriately. When program memory is reloaded prior to resuming user program execution, the modified version of the user program will be the one loaded.

Baud	HR06	HR07
110	93H	04H
150	96H	03H
300	45H	02H
600	9DH	01H
1200	44H	01H
2400	1AH	01H

Table 1. Serial Interface Data Rate Parameters

V. HSE-49 COMMAND DESCRIPTION

Whenever the characters "HSE-49" are present on the system display, a command string may be entered by the operator. In general, all command strings consist of a basic command initiator, an optional command modifier or type-designator, and a number of parameters or delimiters entered as hexadecimal digits. A command is executed, or a command in progress terminated, by pressing the [END/.] key. Logical default values are assumed for the modifier and parameters if either (or both) are omitted. A default parameter assumed for the command modifier will be presented on the display when the first parameter is entered.

Each parameter is a string of up to three hexadecimal digits. If more than three digits are entered, only the most recent three are considered. This allows an erroneous digit to be corrected without respecifying the entire command. A parameter is completed by pressing the [NEXT/.] key. Some commands may only need the

low order part of a parameter; i.e., a command incorporating a data byte (such as [FILL]) will use only the low-order 8 bits of the corresponding parameter; Internal RAM and hardware register addressing uses only seven. In each case, higher order bits are ignored.

A command string is terminated and the command invoked by pressing the [END/.] key. The command will also be invoked by pressing the [NEXT/.] key when no additional parameters are allowed. A command string may be aborted at any point before the command is invoked by pressing the [CLEAR/PREV] key, and the sign-on message will appear.

Errors

An illegal command string, command terminator, or hardware failure will cause an error message and error code number to appear on the display (e.g., "Error-3"). When this occurs, the monitor can be returned to command mode by pressing the [CLEAR] or [END/.] keys. An explanation of the various error codes is given in Appendix D.

Command Classes

Commands for the HSE-49 emulator are divided into general classes, where all commands in each class have the same choice of options or modifiers. A brief description of each command, followed by a description of the allowed options, is presented below by class.

Data Manipulation/Control Command Group

Commands:

[EXAM/CHA]

Display Response — "ECh."

Function — Examine/change memory location.

Causes the memory address specified to be read and presented on the display. New data may be entered (if desired) from the hexadecimal keypad. New data is verified before appearing on the display. Subsequent or previous locations may be read by pressing the [NEXT/.] or [PREV] keys, respectively. Command terminated with [END/.] key.

[FILL]

Display Response — "FIL."

Function — Fill range of memory addresses with a single data value.

Fill the appropriate memory space between the addresses specified by the first two parameters with the low-order byte of the third parameter. If second parameter less than first, only the location specified by the first is affected. If third parameter omitted, zero is assumed. If second and third parameters omitted, individual address specified is cleared. Command is useful for setting a large range of breakpoints; e.g., all of page 3 may be enabled for break with the command:

[FILL][PROG BRK]<300>[,<3FF>][,<1>[.]

[LIST]

Display Response — "LSt."

Function — List memory to output device through HSE-49 serial port.

Display the contents of a range of addresses given by two parameters to a teletype or CRT screen. Data is formatted, 16 separated bytes per line, with the starting address of each line printed. If used with an Intellec® system, the operator first uses ISIS-II to transfer the TTY input to the CRT output ("COPY :TI: TO :CO:") then invokes this command from the keypad. Alternatively, any ISIS device or disk file name(:TO: :LP: :F1:HRDREG.SAV, etc.) may be used as the destination.

[DNLOAD]

Display Response — "dnL."

Function — Download memory through HSE-49 serial port

Load data in hex file format through the serial input port. If used with Intellec® system, the operator first invokes this command from the keypad, then uses ISIS-II to transfer a disk file to the teletype port ("COPY :Fn:file.HEX TO :TO:").

The use of the checksum field for the download command is expanded slightly over the Intel hex file format standard. If the first character of the checksum field is a question mark ("?"), the checksum for that record will not be verified. This allows large object files produced by the assembler to be patched using the ISIS text editor without the necessity of manually recomputing the checksum value.

[UPLOAD]

Display Response — "UPL."

Function — Upload memory through HSE-49 serial port.

Output the contents of a range of addresses specified by the two parameters through the HSE-49 serial port in standard Intel hex file format. If used with Intellec® system, the operator first uses ISIS-II to transfer the TTY input to a disk file ("COPY :TI: TO :Fn:file.HEX"), then invokes this command from the keypad.

Data types allowed:

[PROG MEM]

Display Response — "Pr."

Function — User program memory.

Memory used to develop and execute user program. Addresses 000 through 7FF are the execution processor's memory bank 0; 800 through FFF are memory bank 1.

[REGISTER]

Display Response — "rG."

Function — Register memory and RAM.

Internal RAM of execution processor. Locations 0-7 are working register bank 0; 18-1F are working register bank 1. Only the low-order 7 bits of an address are considered.

[DATA MEM]

Display Response — "dA."

Function — External data memory (if installed).

Memory accessed by execution processor "MOVX A,@Rr" or "MOVX @Rr,A" instructions. High-order 4 bits may or may not be relevant, depending on jumping option selected (explained in Section VII of this note).

[HARD REG]

Display Response — "Hr."

Function — Hardware registers.

The execution processor (EP) hardware registers (accumulator, timer/counter, etc.), as well as several parameters for controlling HSE-49 system status, are accessible through this catch-all memory space. Addresses are as follows:

00 — EP accumulator.

01 — EP PSW.

Bits correspond to 8049 PSW except that bit 3 (unused in the 8049) is used to monitor and alter the state of F1. Bits 2-0 correspond to the stack pointer value after the EP executes a CALL to the mini-monitor; i.e., one greater than when EP was running the user's program.

02 — EP timer/counter.

03 — EP internal RAM location 00.

(This value is also accessible through [REGISTER] space.)

04 — EP program counter (low byte).

05 — EP program counter (high nibble).

06-07 — HSE-49 serial interface baud rate parameters. Defaults to 110 baud; other rates may be selected by loading the values listed in Table 1.

08 — HSE-49 automatic sequencing rate parameter. Used in [GO][AUTO STP] and [GO][AUTO BRK] execution commands. 00 → fastest; FF → slowest. Defaults to 20H; approximately two steps per second.

09 — Monitor version/release number (packed BCD).

0A-0F — Currently unused by the monitor program.

10-7F — Variables used by master processor (MP) monitor. Should not be altered by operator.

[PROG BRK]

Display Response — "Pb."

program execution should halt when running in a mode with breakpoints enabled ([GO][W/ BRK] and [GO][AUTOBRK]). Break will occur if enabled byte is read as the first or last byte of a 2-byte instruction, or read in executing a MOVP, MOVP3, or JMPP instruction. Memory is only 1 bit per location; 00 indicates continue, 01 causes a halt. Addresses 000 through 7FF are the execution processor's memory bank 0; 800 through FFF are memory bank 1.

[DATA BRK]

Display Response — "db."

Function — External data RAM breakpoint memory.

Memory space used to indicate points where data accesses should halt when running in a mode with breakpoints enabled ([GO][W/ BRK] and [GO][AUTOBRK]). Memory is only 1 bit per location; 00 indicates continue, 01 causes a halt. High-order 4 bits of breakpoint address may or may not be relevant, dependent on jumpering option selected for the corresponding data RAM (explained in Section VII of this note).

User Program Execution Control Group

Commands:

[GO]

Display Response — "Go."

Function — Begin execution.

If a parameter is given as part of the command string, execution will begin at that address. Otherwise, the EP program counter (hardware registers 04 and 05) will be used. These will contain the program counter from an earlier program execution break unless they have since been explicitly modified by the operator.

If command is terminated by [END/.], the EP's F1, PSW and stack pointer will be cleared. If command string is terminated by [NEXT/.], PSW will be taken from the EP PSW contents (hardware register 01).

While running the user's program, the characters "—run—" are written on the display. Execution may be halted and another command initiated by pressing the appropriate command key. Execution may be suspended at any time in any mode by pressing the [END/.] key. This will cause the current value of the execution processor program counter and accumulator to appear on the display in the form "PC.234-56". System status is saved in the appropriate hardware registers. At this point, or when an enabled breakpoint is encountered, pressing the [NEXT/.] key will cause the program to continue in the same mode as before. Any other command may be invoked by pressing the appropriate command string.

[GO/RESET]

Display Response — "Gr."

All mnemonics copyrighted © Intel Corporation 1976.

user's program from location 000H. No parameters are allowed. F0, F1, PSW, stack pointer, memory bank flip-flop, etc., are cleared.

Note that this command does not require the use of mini-monitors to initiate program execution. As the last phase of the program development cycle, the 2114 program RAMs and address decoder may be removed and replaced by a ROM or EPROM part (not shown in schematics). This command may be used to start execution when the program RAM has been removed. No interrogation of EP status or internal RAM may be done, nor are break or single-step modes allowed in this case, though the 2102A breakpoint RAM outputs may still be used to trigger a logic analyzer.

Execution modes allowed:

[NO BRK]

Display Response — "nb."

Function — Without breakpoints.

Full-speed execution without breakpoints enabled. Does not affect the state of the breakpoint memories.

[SING STP]

Display Response — "SSt."

Function — Single Step.

Step through program one instruction at a time. After each instruction is executed, execution halts with the current value of the Execution Processor Program Counter and Accumulator appearing on the display in the form "PC.234-56". System status is saved in the appropriate Hardware Registers. At the point, [NEXT/.] will cause the program to execute one more instruction, or any other command may be invoked by pressing the appropriate command string. Does not affect the state of the Breakpoint Memories.

[W/ BRK]

Display Response — "br."

Function — With breakpoints.

Full-speed execution with breakpoints enabled. When a breakpoint is encountered, execution halts with the current value of the execution processor program counter and accumulator appearing on the display in the form "PC.234-56". System status is saved in the appropriate hardware registers. At this point, [NEXT/.] will cause the program to continue until the next breakpoint is reached, or any other command may be invoked by pressing the appropriate command string.

[AUTO STP]

Display Response — "ASt."

Function — Automatically sequence through a series of instructions.

Step through program one instruction at a time. After each instruction is executed, execution halts with the current value of the execution processor program counter and accumulator appearing on the display in the form "PC.234-56". System status is saved in the appropriate hardware registers. Execution resumes after a time determined by contents of hardware register 08. Does not affect the state of the breakpoint memories.

[AUTO BRK]

Display Response — "Abr."

Function — Automatically sequence between breakpoints.

Execute a series of instructions in real time between breakpoints. When breakpoint is encountered, halt EP temporarily while program counter and accumulator contents are displayed, then continue. Display is sustained after execution resumes. Does not affect the state of the breakpoint memories.

Breakpoint Control Command Group

Commands:

[B]

Display Response — "Stb."

Function — Breakpoint set.

Set breakpoint for the address given. Multiple breakpoints may be set by entering additional addresses, separated by the [NEXT/] key. Command terminated by pressing [END/]. Action taken is to fill the appropriate breakpoint memory locations with logical ones.

[C]

Display Response — "CLb."

Function — Clear breakpoint.

Clear breakpoint for the address given. Multiple breakpoints may be cleared by entering additional addresses, separated by the [NEXT/] key. Command terminated by pressing [END/]. Action taken is to fill the appropriate breakpoint memory locations with logical zeroes.

Data types allowed:

[PROG MEM]

Display Response — "Pr."

Function — Break on program memory fetch.

Applies command to the program breakpoint memory space.

[DATA MEM]

Display Response — "dA."

Function — Break on data memory access.

Applies command to the external data breakpoint memory space.

System Control Command Group

Command:

[SYS RST]

Display Response — "HSE-49."

Function — System reset.

Reset both the MP and EP and clear all breakpoints (requires approximately one second). CAUTION — If reset while EP is executing the user's program, the low order section of program memory (about 23 bytes) will be altered.

VI. SYSTEM LIMITATIONS

In designing the HSE-49 emulator, certain compromises were made in an attempt to maximize the usefulness of the emulator while keeping the circuitry simple and inexpensive. As a result, the following limitations exist and must be taken into account when using the system.

1. As explained in Section IV, user program execution is terminated (by single-stepping, breakpoints, pressing the [END/] key, etc.) by forcing the execution processor to execute a "CALL" instruction to the mini-monitor. This uses one level of the EP subroutine stack. The EP PSW reflects the value of the stack pointer *after* processing this CALL. As a result, the value indicated for stack depth by examining the EP PSW (hardware register 01) is one greater than the depth when the break was initiated. The user program must not be using all eight levels of stack when a break is initiated or the bottom level will be destroyed.
2. User program is initiated (by the [GO] command or when resuming execution after a breakpoint, single-stepping, etc.) by forcing the EP to execute an "RETR" instruction. This will clear the EP interrupt-in-progress flip-flop. If the user program allows both external and timer interrupts to be enabled at the same time, care must be taken to avoid causing a break while the EP is within an interrupt servicing routine. No limitation is placed on breakpoints or single-stepping in the background program because of this.
3. When the user program execution is terminated (by a break, single-stepping, etc.) and later resumed, the EP timer/counter is restored to its value when the break occurred (unless modified by the user). The prescaler, however, will have changed. Thus, up to 31 machine cycles may be "lost" or "gained" if a break occurs while the timer is running.
4. Timer interrupts occurring at the same time as an EP break may be ignored if the timer overflow occurs after breaking user program execution before the timer value is saved.
5. The 8049 "RET" and "RETR" instructions are each 1-byte, 2-cycle instructions. During the second cycle the byte following the return instruction is fetched and ignored. If a program breakpoint is set for a location following a "RET" or "RETR" instruction, a break will be initiated when the return is executed.

6. Breakpoints should not be placed in the last 3 bytes of an EP memory bank (locations 7FDH-7FFH and 0FFDH-0FFFH). User program should not be single-stepped or auto-stepped through these locations.
7. Since I/O configuration is determined by external hardware rather than software, I/O modes may not be altered while a program is executing. (See Section VII for further details.)
8. The "ANL BUS,#nn" and "ORL BUS,#nn" instructions may not be used in the user program, as external hardware cannot properly restore these functions.
9. The memory bank select flag is not affected by the user program break sequence. Upon resuming execution with the [GO] command this flag will remain in the same state as before the preceding break. The flag may be cleared only by executing the [GO/RESET] or [SYS RST] commands.

VII. HARDWARE CONFIGURATIONS

A number of control and status lines are available to the user. All are low-power Schottky TTL-compatible signals.

TP1 — Unused MP input.

TP2 — Unused MP output.

TP3 — User program suspended. Low when EP running user code. High when halted or running mini-monitors.

TP4 — Breakpoint encountered. Normally low. High-level pulse generated when breakpoint passed. Useful for triggering logic analyzers, oscilloscopes, etc.

TP5 & TP6 — Memory matrix mode control. Select program vs. data RAM, link mapping configuration, etc. (See Appendix B for details.)

TP7 — Bus control. Low when MP controls common memory buses. High when EP controls memory buses.

The HSE-49 emulator hardware is designed to allow the user to reconfigure the system for a wide variety of different applications by installing or removing jumper wires or additional components. The schematics in Appendix A show the components needed for a variety of different configurations. In general, not all of the devices are required (or allowed) for any one configuration. The devices which are required are included in the following description.

The types of options allowed are divided below into several general classes and subdivided into mutually-independent features. Within some of these features there are numbered, mutually exclusive configurations; i.e., the serial interface (if desired) may use either

current-loop or RS-232C current buffers, but not both at one time.

Standard Operating Configuration

(Minimum system configurations — up to 4K program RAM; no data RAM; no serial interfaces; no execution processor I/O reconstruction.)

A. Basic 2K monitor from Appendix B:

Install resistors R4-R6
Install transistor Q1
Install crystals Y1-Y2
Install capacitors C5-C38
Install switches S1-S33
Install displays DS1-DS8
Install IC1-IC2
Install RP3-RP5
Install IC6-IC7
Install RP8
Install IC9
Install IC15-IC20
Install IC25-IC30
Install IC34
Install IC36-IC38
Install A1-A2
Install B1-B2
Install C1-C3
Install jumpers 13-15
Install jumpers 17-18
Install jumper 20

B. Expansion 2K monitor:

Install IC14
Remove jumper 17

Serial Interface Buffer Selection

A. Current loop serial interfaces (4N46s) installed for use with full Inteltec® Model 800 development system TTY port.

Install IC21-IC22
Install resistor R1-R3
Install jumpers 4-9
(Remove RS-232 jumpers)

B. RS-232C serial interfaces (MC1488 and MC1489) installed for use with CRT as output device for data dumps:

Install IC23-IC24
Install jumpers 1-3
Install jumpers 10-11
(Remove current-loop jumpers)

External Data RAM Address Decoding Scheme for Execution Processor

A. Up to 16 pages of on-board external data RAM installed for execution processor (addresses 0 through

0FFFH = 4K bytes); port 2 used for addressing pages 0 through 15:

Install jumpers 21-25
Install jumper 27
Install A5-A8
Install B5-B8
Install C5-C8

- B. One page of on-board external data RAM installed for execution processor (addresses 0 through 0FFFH); port 2 not used for data addressing:

Install jumper 26
Install jumper 28
Install A5
Install B5
Install C5

Connect the outputs of IC20, pins 7, 9, 10, & 11 to the inputs of a 74LS21 AND gate (not shown). Connect the output to CE and CS inputs of A5-C5. (Note: these signals are all present at jumpers 21-24 on the schematics.)

Reconstructing I/O for Execution Processor

- A. Application of port 2, pins P23-P20:

- (1) Using P23-P20 for latched output data (used with "OUTL P2,A", "ANL P2,#data", and "ORL P2,#data" instructions):

Install IC31

- (2) Using P23-P20 for interfacing to an 8243 in user's prototype:

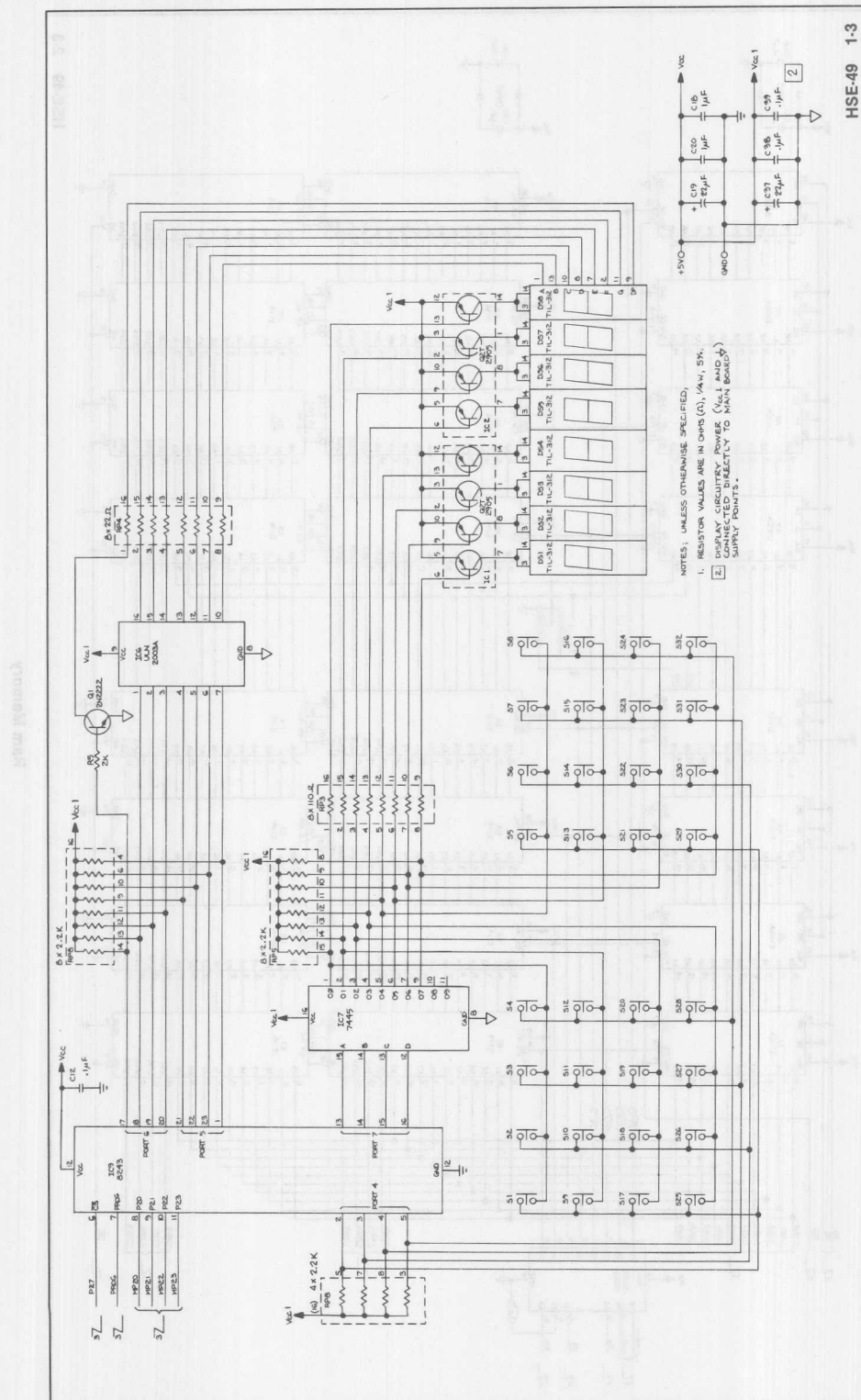
Connect D3-D0 pins on IC31 socket to corresponding Q3-Q0 pins.

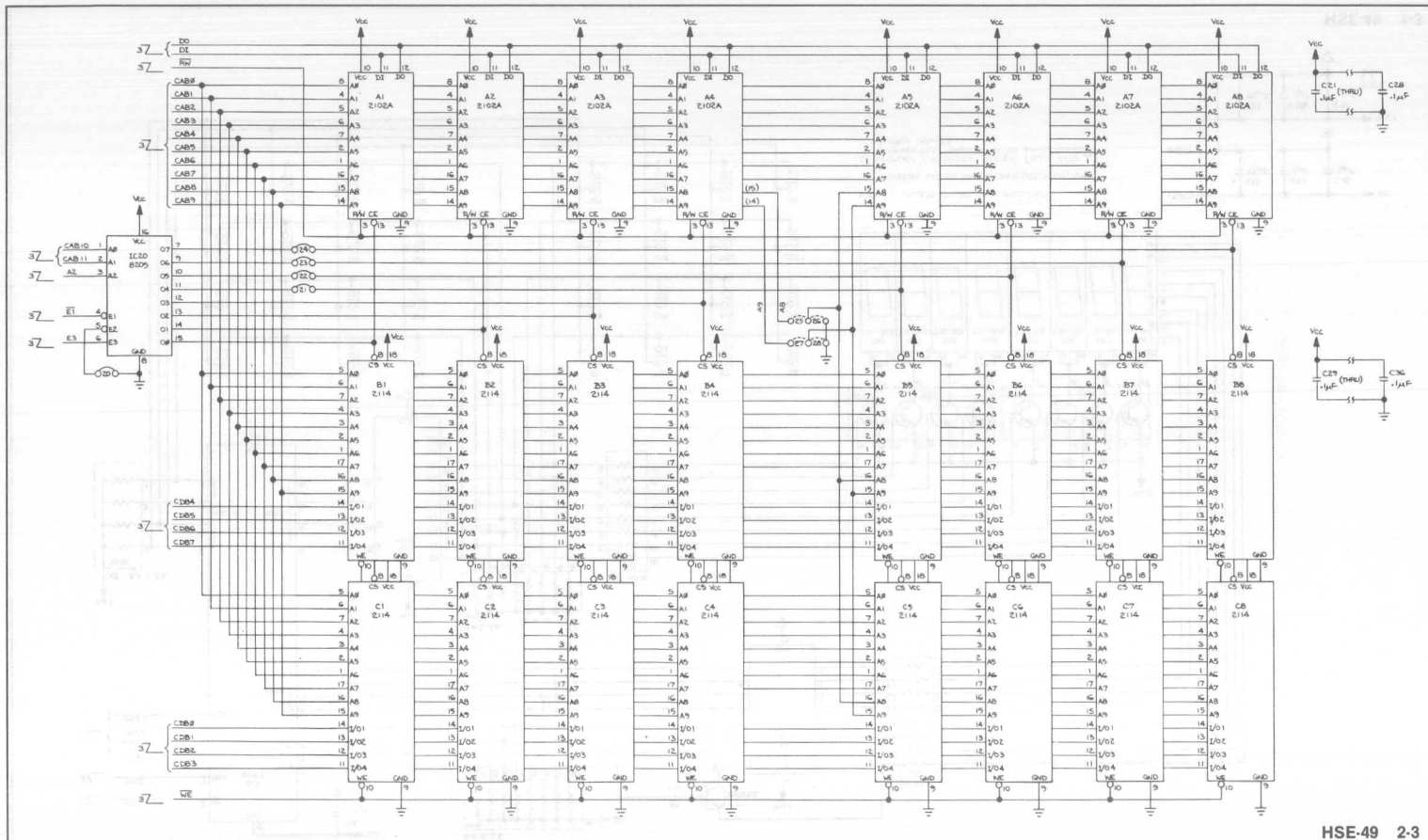
- B. Application of execution processor BUS:

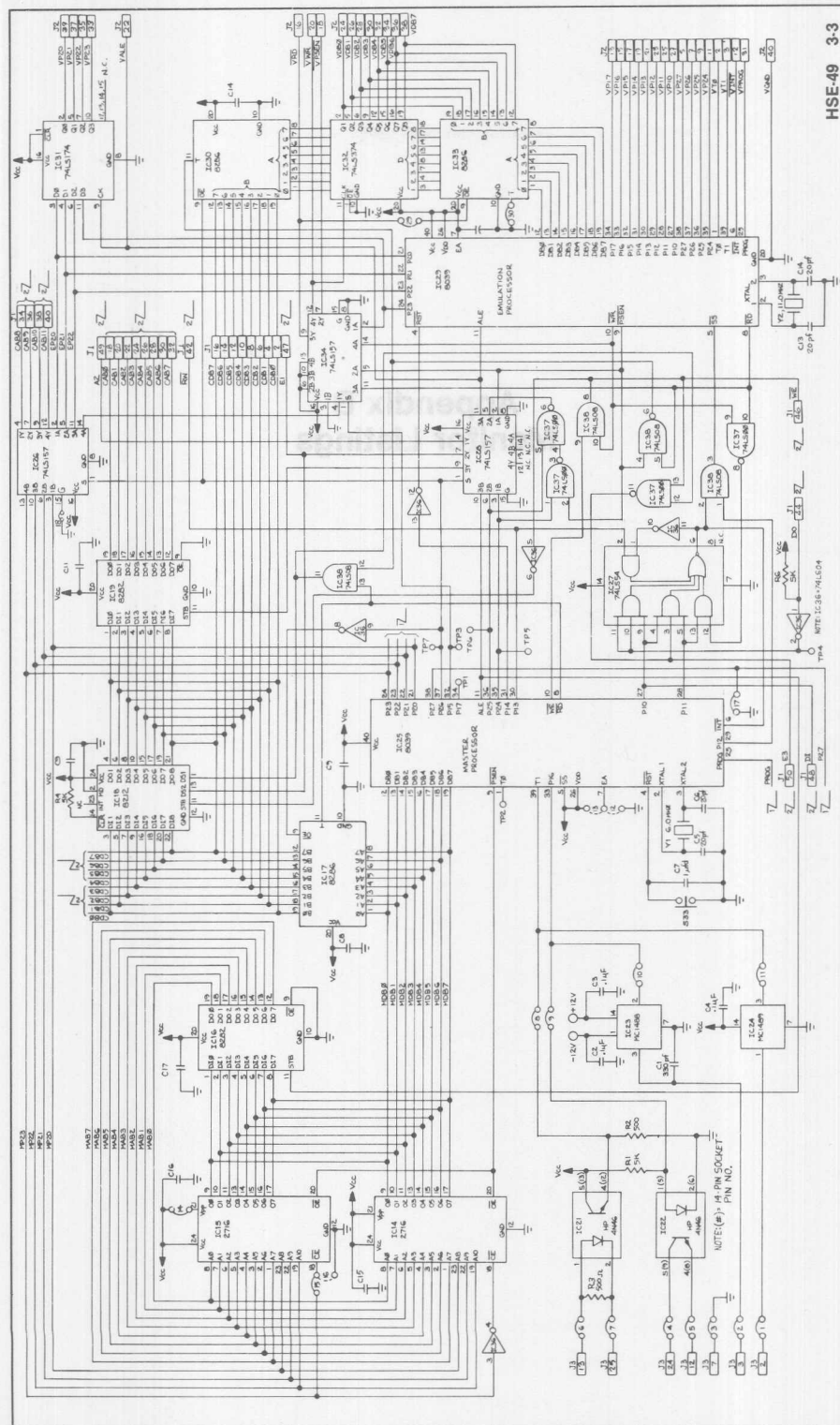
- (1) Use of BUS as latched output port ("OUTL BUS,A"):

Install IC32

Appendix A Schematic Diagrams









Appendix B Monitor Listings

ISIS-II MCS-48/UPI-41 MACRO ASSEMBLER, V3.0
HSE-49(TM) EMULATOR MONITOR VERSION 2.5

PAGE 1

LOC	OBJ	LINE	SOURCE STATEMENT
		1	\$MACROFILE NOGEN NOCOND XREF
		2	\$TITLE('HSE-49(TM) EMULATOR MONITOR VERSION 2.5')
		3	;
		4	*****
		5	;
		6	PROGRAM: HSE-49(TM) EMULATOR MONITOR
		7	VERS 2.5/789
		8	;
		9	COPYRIGHT (C) 1979
		10	INTEL CORPORATION
		11	3065 BOWERS AVENUE
		12	SANTA CLARA, CALIFORNIA 95051
		13	;
		14	*****
		15	;
		16	ABSTRACT
		17	=====
		18	;
		19	THIS PROGRAM CONTAINS THE SOFTWARE NECESSARY TO RUN THE HSE-49(TM)
		20	HIGH-SPEED EMULATOR FOR INTEL'S MCS-48(TM) FAMILY FAMILY OF MICROCOMPUTERS.
		21	THE EMULATOR PROVIDES AN ASSORTMENT OF UTILITY FUNCTIONS FOR
		22	DEVELOPING AND DEBUGGING 8048-BASED APPLICATIONS, INCLUDING THE
		23	ABILITY TO ENTER AND MODIFY PROGRAMS IN PROGRAM RAM.
		24	ALTER DATA, SINGLE-STEP SECTIONS OF A PROGRAM, AND EXECUTE PROGRAMS
		25	AT SPEEDS OF UP TO 11 MHZ, WITH OR WITHOUT BREAKPOINTS ENABLED.
		26	THE EMULATOR IS DESCRIBED IN GREATER DEPTH IN INTEL'S APPLICATION NOTE
		27	AP-55, "A HIGH-SPEED EMULATOR FOR INTEL MCS-48(TM) MICROCOMPUTERS."
		28	;
		29	PROGRAM ORGANIZATION
		30	=====
		31	;
		32	THIS LISTING IS ORGANIZED AS FOLLOWS:
		33	;
		34	INTRODUCTION AND HARDWARE OVERVIEW;
		35	VARIABLE DECLARATION AND DEFINITION;
		36	POWER-ON SYSTEM INITIALIZATION;
		37	KEYBOARD COMMAND PARSER AND ASSOCIATED TABLES;
		38	IMPLEMENTATIONS OF THE PRIMARY COMMANDS;
		39	DATA ACCESSING UTILITY SUBROUTINES USED THROUGHOUT;
		40	KEYBOARD SCANNING AND DISPLAY DRIVING SUBROUTINE;
		41	KEYBOARD AND DISPLAY INTERFACING UTILITIES;
		42	ROUTINES AND UTILITY SUBROUTINES WHICH INTERACT BETWEEN MP AND EP.
		43	;
		44	;
		45	\$EJECT

LOC	OBJ	LINE	SOURCE STATEMENT
		46 ;	
		47 ;	INTRODUCTION AND HARDWARE OVERVIEW
		48 ;	=====
		49 ;	
		50 ;	THE EMULATOR DESIGN USES TWO MICROPROCESSORS. ONE PROCESSOR CONTROLS
		51 ;	SYSTEM STATUS, INTERPRETS MONITOR COMMANDS, AND COMMUNICATES
		52 ;	WITH THE OUTSIDE WORLD THROUGH THE ON-BOARD KEYBOARD, DISPLAY, SERIAL
		53 ;	INTERFACES, CONTROL SIGNALS, ETC.
		54 ;	A SECOND PROCESSOR IS USED TO ACTUALLY
		55 ;	EXECUTE THE USER'S PROGRAM UNDER THE CONTROL OF THE FIRST.
		56 ;	THESE PROCESSORS ARE REFERRED TO
		57 ;	THROUGHOUT THIS PROGRAM AS THE MASTER PROCESSOR (MP) AND EXECUTION
		58 ;	PROCESSOR (EP) RESPECTIVELY.
		59 ;	
		60 ;	THE PROGRAM IN THIS LISTING IS EXECUTED BY THE MASTER PROCESSOR.
		61 ;	AT THE END OF THIS LISTING ARE SEVERAL SHORT "MINI-MONITOR OVERLAYS"
		62 ;	WHICH THE EXECUTION PROCESSOR EXECUTES WHEN INTERACTION BETWEEN THE
		63 ;	TWO PROCESSORS IS NECESSARY.
		64 ;	
		65 ;	THIS PROGRAM WAS WRITTEN USING A NUMBER OF MACROS TO HANDLE THE ALLOCATION
		66 ;	OF MPU RESOURCES (WORKING REGISTERS, INTERNAL RAM, AND MP MONITOR ROM
		67 ;	FOR CODE AND DATA STORAGE). THESE MACRO DEFINITIONS ARE INCLUDED IN A FILE
		68 ;	NAMED "ALLOC.MAC," AND ARE PRINTED IN THIS LISTING FOR REFERENCE.
		69 ;	ANOTHER SET OF MACROS IS USED TO SIMPLIFY THE ACCESSING OF VARIABLES
		70 ;	STORED IN INTERNAL RAM (AS OPPOSED TO WORKING REGISTERS) BY USING R1 TO
		71 ;	INDIRECTLY ADDRESS THE APPROPRIATE RAM LOCATION WHEN NECESSARY.
		72 ;	THESE MACROS ARE INCLUDED IN "MOPCOD.MAC", AND ARE ALSO PRINTED HERE.
		73 ;	COMPLETE UNDERSTANDING OF THESE MACROS IS NOT REQUIRED TO UNDERSTAND THE
		74 ;	MONITOR PROPER. ALL LINES WHICH ACTUALLY PRODUCE OBJECT CODE APPEAR IN
		75 ;	THE LISTING ITSELF, INDENTED TWO SPACES FROM THE NORMAL TABULATION COLUMNS.
		76 ;	THE ACTUAL MONITOR PROGRAM FOR THE EMULATOR BEGINS AT APPROXIMATELY
		77 ;	SOURCE LINE NUMBER 500.
		78 ;	
		79 ;	Lines generated by macro expansion are flagged by a plus sign ("+")
		80 ;	immediately following the source line number.
		81 ;	A number of lines from the various macro definitions which do not
		82 ;	produce any object code are processed by the assembler
		83 ;	as these macros are expanded. When this is the case, these lines are
		84 ;	suppressed from the list file. As a result, the line numbers are
		85 ;	not always consecutive where a macro is being invoked.
		86 ;	
		87 ;	NOTE:
		88 ;	====
		89 ;	"SOURCE-LINE" REFERS TO THE DECIMAL NUMBERS LEFT OF EACH INSTRUCTION.
		90 ;	AT THE END OF THE LISTING IS AN ASSEMBLY CROSS-REFERENCE TABLE INDICATING
		91 ;	THE SEQUENTIAL SOURCE-LINE NUMBER OF ALL INSTANCES WHERE ANY VARIABLE
		92 ;	IS DEFINED OR REFERENCED. THIS WILL BE OF GREAT ASSISTANCE IN
		93 ;	LOCATING SPECIFIC SUBROUTINES, ETC. IN THE LISTING.
		94 ;	
		95 ;	MNEMONICS COPYRIGHT (C) 1976 INTEL CORPORATION
		96 ;	
		97	\$EJECT

LOC	OBJ	LINE	SOURCE STATEMENT	ASSEMBLY SOURCE	LINE	LOC	OBJ
		98 \$	INCLUDE(:F0:ALLOC.MAC)				
0000		= 99 ?R1	SET 0				
		= 100 ;					
0000		= 101 ?R00	EQU 0				
0001		= 102 ?R01	EQU 1				
0002		= 103 ?RAM	EQU 2				
0003		= 104 ?CONST	EQU 3				
0004		= 105 ?R	EQU 4 ; ACCUMULATOR VARIABLE TYPE				
		= 106 ;					
		= 107 ;	THE FOLLOWING INITIALIZES THE LINKED LIST POINTERS FOR				
		= 108 ;	THE REGISTER ALLOCATION AND DEALLOCATION ROUTINES.				
		= 109 ;					
0003		= 110 ?B0R2	SET 3				
0004		= 111 ?B0R3	SET 4				
0005		= 112 ?B0R4	SET 5				
0006		= 113 ?B0R5	SET 6				
0007		= 114 ?B0R6	SET 7				
0008		= 115 ?B0R7	SET 8				
		= 116 ;					
0002		= 117 ?B0PNT	SET 2				
		= 118 ;					
0003		= 119 ?B1R2	SET 3				
0004		= 120 ?B1R3	SET 4				
0005		= 121 ?B1R4	SET 5				
0006		= 122 ?B1R5	SET 6				
0007		= 123 ?B1R6	SET 7				
0008		= 124 ?B1R7	SET 8				
		= 125 ;					
0002		= 126 ?B1PNT	SET 2				
		= 127 ;					
0000		= 128 ORPG0	SET 000H				
0100		= 129 ORPG1	SET 100H				
0200		= 130 ORPG2	SET 200H				
0300		= 131 ORPG3	SET 300H				
0400		= 132 ORPG4	SET 400H				
0500		= 133 ORPG5	SET 500H				
0600		= 134 ORPG6	SET 600H				
0700		= 135 ORPG7	SET 700H				
		= 136 ;					
		= 137 \$EJECT					

	= 140 ;	START OF ALLOCATION MACROS				
	= 141 ;					
	= 142 ;	*****				
	= 143 ;					
	= 144 ?RSAVE	MACRO SYMBOL, BANK, PNTVAL				
-	= 145 IF	PNTVAL EQ 0				
-	= 146	ERROR 2				
-	= 147	EXITM				
-	= 148 ENDIF					
-	= 149 \$	SAVE GEN				
-	= 150	SYMBOL SET R&PNTVAL				
-	= 151 \$	RESTORE				
-	= 152 ?B&BANK&PNT	SET ?B&BANK&R&PNTVAL				
	= 153	ENDM				
	= 154 ;					
	= 155 ;					
0020	= 156 ?MINDX	SET 20H				
	= 157 ;					
	= 158 ?MSAVE	MACRO SYMBOL, LENGTH, ADDR				
	= 159 \$	SAVE GEN				
-	= 160	SYMBOL EQU ADDR				
-	= 161 \$	RESTORE				
-	= 162 ?MINDX	SET ?MINDX+LENGTH				
	= 163 ENDM					
	= 164 ;					
	= 165 MBLOCK	MACRO SYMBOL, LENGTH				
-	= 166 ?&SYMBOL	EQU 3				
-	= 167	?MSAVE SYMBOL, LENGTH, ?MINDX				
	= 168 ENDM					
	= 169 ;					
	= 170 DECLARE	MACRO SYMBOL, TYPE				
-	= 171 ?&SYMBOL	SET ?&TYPE				
-	= 172 IF	?&TYPE EQ 2				
-	= 173	?MSAVE SYMBOL, 1, ?MINDX				
-	= 174	EXITM				
-	= 175 ENDIF					
-	= 176 IF	?&TYPE EQ 0				
-	= 177	?RSAVE SYMBOL, 0, ?B&PNT				
-	= 178	EXITM				
-	= 179 ENDIF					
-	= 180 IF	?&TYPE EQ 1				
-	= 181	?RSAVE SYMBOL, 1, ?B&PNT				
-	= 182	EXITM				
-	= 183 ENDIF					
	= 184	ENDM				
	= 185 ;					
	= 186 \$	EJECT				

LOC	OBJ	LINE	SOURCE STATEMENT	INSTRUMENT INDEX	VAL	ADD	OBJ
		= 187 ;					
		= 188 ;REORG	MACRO TO RESET THE INSTRUCTION LOCATION COUNTER				
		= 189 ;	TO THE FIRST FREE LOCATION ON THE FIRST PAGE MODULE WILL				
		= 190 ;	FIT WITHIN				
		= 191 REORG	MACRO LOCATION				
		= 192 \$SAVE GEN					
-		= 193	ORG LOCATION				
-		= 194 \$RESTORE					
-		= 195	ENDM				
-		= 196 ;					
-		= 197 ;CODEBLK	MACRO TO FIND A PAGE OF ROM				
-		= 198 ;	WHICH THIS BLOCK OF CODE WILL FIT WITHIN				
-		= 199 CODEBLK	MACRO LENGTH				
-		= 200 ?LENGTH	SET LENGTH				
-		= 201 IF	HIGH(ORPG0+LENGTH-1) EQ 0				
-		= 202	REORG %ORPG0				
-		= 203 ?START	SET \$				
-		= 204 EXITM					
-		= 205 ENDIF					
-		= 206 IF	HIGH(ORPG1+LENGTH-1) EQ 1				
-		= 207	REORG %ORPG1				
-		= 208 ?START	SET \$				
-		= 209 EXITM					
-		= 210 ENDIF					
-		= 211 IF	HIGH(ORPG2+LENGTH-1) EQ 2				
-		= 212	REORG %ORPG2				
-		= 213 ?START	SET \$				
-		= 214 EXITM					
-		= 215 ENDIF					
-		= 216 IF	HIGH(ORPG4+LENGTH-1) EQ 4				
-		= 217	REORG %ORPG4				
-		= 218 ?START	SET \$				
-		= 219 EXITM					
-		= 220 ENDIF					
-		= 221 IF	HIGH(ORPG5+LENGTH-1) EQ 5				
-		= 222	REORG %ORPG5				
-		= 223 ?START	SET \$				
-		= 224 EXITM					
-		= 225 ENDIF					
-		= 226 IF	HIGH(ORPG6+LENGTH-1) EQ 6				
-		= 227	REORG %ORPG6				
-		= 228 ?START	SET \$				
-		= 229 EXITM					
-		= 230 ENDIF					
-		= 231 IF	HIGH(ORPG7+LENGTH-1) EQ 7				
-		= 232	REORG %ORPG7				
-		= 233 ?START	SET \$				
-		= 234 EXITM					
-		= 235 ENDIF					
-		= 236 IF	HIGH(ORPG3+LENGTH-1) EQ 3				
-		= 237	REORG %ORPG3				
-		= 238 ?START	SET \$				
-		= 239 EXITM					
-		= 240 ENDIF					
-		= 241	ERROR 0 ;** INSUFFICIENT SPACE FOR CODE ON ANY PAGE **				

LOC	OBJ	LINE	SOURCE STATEMENT	TEMPORARY SYMBOLS	FILE	140	300
		= 242	ENDM				
		= 243	;DATBLK INSERTS ONTO PAGE 3				
		= 244	DATBLK MACRO LENGTH				
-		= 245	?LENGTH SET LENGTH				
-		= 246	IF HIGH(ORGP63+LENGTH-1) EQ 3				
-		= 247	REORG %ORGP63				
-		= 248	?START SET \$				
-		= 249	EXITM				
-		= 250	ENDIF				
-		= 251	ERROR 0 ;*** INSUFFICIENT SPACE FOR DATA BLOCK ON PAGE 3 ***				
-		= 252	ENDM				
-		= 253	?SIZE PRINTS A LINE TO THE SOURCE FILE GIVING BLOCK SIZE.				
-		= 254	; AND UPDATES APPROPRIATE ORGP#				
-		= 255	?SIZE MACRO BLK,PGE				
-		= 256	\$SAVE GEN				
-		= 257	SIZE SET BLK				
-		= 258	;				
-		= 259	*****				
-		= 260	IF ?LENGTH LT SIZE				
-		= 261	ERROR 0 ;*** SIZE EXCEEDS SPACE CHECKED FOR BY CODEBLK MACRO				
-		= 262	ENDIF				
-		= 263	IF HIGH(\$-1) NE HIGH(?START)				
-		= 264	ERROR 0 ;*** CODE OR DATA BLOCK ROLLED OVER PAGE BOUNDARY ***				
-		= 265	ENDIF				
-		= 266	\$RESTORE				
-		= 267	ORGP63PGE SET \$				
-		= 268	ENDM				
-		= 269	;SIZECHK CHECKS SIZE OF PRECEDING BLOCK, PRINTS SIZE TO .LS1 FILE.				
-		= 270	SIZECHK MACRO				
-		= 271	?SIZE %(\$-?START),%HIGH(?START)				
-		= 272	ENDM				
-		= 273	;				
-		= 274	;				
-		= 275	;RSOURCE CODE SPACE ALLOCATION SUMMARY STATEMENT				
-		= 276	RSOURCE MACRO				
-		= 277	\$SAVE LIST GEN				
-		= 278	PGSIZE SET ORGP60-000H ;BYTES USED ON PAGE 0				
-		= 279	PGSIZE SET ORGP61-100H ;BYTES USED ON PAGE 1				
-		= 280	PGSIZE SET ORGP62-200H ;BYTES USED ON PAGE 2				
-		= 281	PGSIZE SET ORGP63-300H ;BYTES USED ON PAGE 3				
-		= 282	PGSIZE SET ORGP64-400H ;BYTES USED ON PAGE 4				
-		= 283	PGSIZE SET ORGP65-500H ;BYTES USED ON PAGE 5				
-		= 284	PGSIZE SET ORGP66-600H ;BYTES USED ON PAGE 6				
-		= 285	PGSIZE SET ORGP67-700H ;BYTES USED ON PAGE 7				
-		= 286	\$EJECT				
-		= 287	\$RESTORE				
-		= 288	ENDM				
-		= 289	\$EJECT				
			*** END OF PAGE ***				

LOC	OBJ	LINE	SOURCE STATEMENT	INSTRUCTIONS	SIZE	ADDRESS
		290 ;				
		291 \$	INCLUDE(:F0:MOPCOD.MAC)			
		= 292 ;				
		= 293 ;?FORM1 MACRO	FOR GENERALIZING OPCODE INSTRUCTION			
		= 294 ;				
		= 295 ?FORM1 MACRO	OPCODE, SRC			
-		= 296 IF	?&SRC EQ 2			
-		= 297 \$	SAVE GEN			
-		= 298	MOV R1, #SRC			
-		= 299	OPCODE A, @R1			
-		= 300 \$	RESTORE			
-		= 301	EXITM			
-		= 302 ENDIF				
-		= 303 IF	?&SRC EQ 0 OR ?&SRC EQ 1			
-		= 304 \$	SAVE GEN			
-		= 305	OPCODE A, SRC			
-		= 306 \$	RESTORE			
-		= 307	EXITM			
-		= 308 ENDIF				
-		= 309 IF	?&SRC EQ 3			
-		= 310 \$	SAVE GEN			
-		= 311	OPCODE A, #SRC			
-		= 312 \$	RESTORE			
-		= 313	EXITM			
-		= 314 ENDIF				
-		= 315	ERROR 1			
-		= 316 ENDM				
-		= 317 ;				
-		= 318 ;?FORM2 MACRO	FOR GENERALIZING MOVES FROM THE ACC TO A VARIABLE			
-		= 319 ?FORM2 MACRO	DEST			
-		= 320 IF	?&DEST EQ 2			
-		= 321 \$	SAVE GEN			
-		= 322	MOV R1, #DEST			
-		= 323	MOV @R1, A			
-		= 324 \$	RESTORE			
-		= 325	EXITM			
-		= 326 ENDIF				
-		= 327 IF	?&DEST EQ 0 OR ?&DEST EQ 1			
-		= 328 \$	SAVE GEN			
-		= 329	MOV DEST, A			
-		= 330 \$	RESTORE			
-		= 331	EXITM			
-		= 332 ENDIF				
-		= 333	ERROR 1			
-		= 334 ENDM				
-		= 335 ;				
-		= 336 ;?FORM3 MACRO	FOR GENERALIZING MOVES FROM THE ACC TO A VARIABLE			
-		= 337 ;	WHEN IT IS KNOWN THAT R1 (IF NEEDED FOR INDIRECT ADDRESSING)			
-		= 338 ;	IS ALREADY PRESET.			
-		= 339 ?FORM3 MACRO	DEST			
-		= 340 IF	?&DEST EQ 2			
-		= 341 \$	SAVE GEN			
-		= 342	MOV @R1, A			
-		= 343 \$	RESTORE			
-		= 344	EXITM			

```

= 346 IF      ?&DEST EQ 0 OR ?&DEST EQ 1
= 347 $      SAVE GEN
= 348      MOV      DEST, A
= 349 $      RESTORE
= 350      EXITM
= 351 ENDIF
= 352      ERROR 1
= 353 ENDM
= 354 ;
= 355 ;?FORM4 MACRO FOR GENERALIZING 'MOV A, SRC' INSTRUCTION
= 356 ?FORM4 MACRO SRC
= 357 IF      ?&SRC EQ 2
= 358 $      SAVE GEN
= 359      MOV      R1, #SRC
= 360      MOV      A, @R1
= 361 $      RESTORE
= 362      EXITM
= 363 ENDIF
= 364 IF      ?&SRC EQ 0 OR ?&SRC EQ 1
= 365 $      SAVE GEN
= 366      MOV      A, SRC
= 367 $      RESTORE
= 368      EXITM
= 369 ENDIF
= 370 IF      ?&SRC EQ 3
= 371 $      SAVE GEN
= 372      MOV      A, #SRC
= 373 $      RESTORE
= 374      EXITM
= 375 ENDIF
= 376      ERROR 1
= 377 ENDM
= 378 ;
= 379 ;?FORM5 MACRO FOR GENERALIZING MOVING A CONSTANT INTO A VARIABLE
= 380 ?FORM5 MACRO DEST, CONST
= 381 IF      ?&DEST EQ 0 OR ?&DEST EQ 1 OR ?&DEST EQ 4
= 382 $      SAVE GEN
= 383      MOV      DEST, #CONST
= 384 $      RESTORE
= 385      EXITM
= 386 ENDIF
= 387 IF      ?&DEST EQ 2
= 388 $      SAVE GEN
= 389      MOV      R1, #DEST
= 390      MOV      @R1, #CONST
= 391 $      RESTORE
= 392      EXITM
= 393 ENDIF
= 394      ERROR 1
= 395 ENDM
= 396 ;
= 397 ;MMOV MACRO GENERALIZED MOVE FROM SRC TO DEST
= 398 MMOV MACRO DEST, SRC
= 399 IF      ?&SRC EQ 3

```

LOC	OBJ	LINE	SOURCE STATEMENT	SYMBOLS	VALUES	LOC	OBJ
-		= 400	?FORM5 DEST, SRC	00000000 00000000 00000000 00000000	00000000		
-		= 401	EXITM	00000000 00000000 00000000 00000000	00000000		
-		= 402	ENDIF				
-		= 403	IF ?&DEST EQ 4				
-		= 404	?FORM1 MOV, SRC	00000000 00000000 00000000 00000000	00000000		
-		= 405	EXITM	00000000 00000000 00000000 00000000	00000000		
-		= 406	ENDIF				
-		= 407	IF ?&SRC EQ 4				
-		= 408	?FORM2 DEST	00000000 00000000 00000000 00000000	00000000		
-		= 409	EXITM	00000000 00000000 00000000 00000000	00000000		
-		= 410	ENDIF				
-		= 411	?FORM1 MOV, SRC	00000000 00000000 00000000 00000000	00000000		
-		= 412	?FORM2 DEST	00000000 00000000 00000000 00000000	00000000		
-		= 413	ENDM				
-		= 414	?BINOP MACRO GENERALIZES ARITHMETIC AND LOGICAL OPERATIONS				
-		= 415	?BINOP MACRO OPCODE, DEST, SRC				
-		= 416	IF ?&DEST EQ 4				
-		= 417	?FORM1 OPCODE, SRC	00000000 00000000 00000000 00000000	00000000		
-		= 418	EXITM	00000000 00000000 00000000 00000000	00000000		
-		= 419	ENDIF				
-		= 420	IF ?&SRC EQ 4				
-		= 421	?FORM1 OPCODE, DEST	00000000 00000000 00000000 00000000	00000000		
-		= 422	?FORM3 DEST	00000000 00000000 00000000 00000000	00000000		
-		= 423	EXITM	00000000 00000000 00000000 00000000	00000000		
-		= 424	ENDIF				
-		= 425	?FORM1 MOV, SRC	00000000 00000000 00000000 00000000	00000000		
-		= 426	?FORM1 OPCODE, DEST	00000000 00000000 00000000 00000000	00000000		
-		= 427	?FORM3 DEST	00000000 00000000 00000000 00000000	00000000		
-		= 428	ENDM				
-		= 429	MADD MACRO FOR GENERALIZING ADD INSTRUCTION				
-		= 430	MADD MACRO DEST, SRC				
-		= 431	?BINOP ADD, DEST, SRC	00000000 00000000 00000000 00000000	00000000		
-		= 432	ENDM				
-		= 433	;				
-		= 434	MADDC MACRO FOR GENERALIZING ADDC INSTRUCTION				
-		= 435	MADDC MACRO DEST, SRC				
-		= 436	?BINOP ADDC, DEST, SRC	00000000 00000000 00000000 00000000	00000000		
-		= 437	ENDM				
-		= 438	;				
-		= 439	MANL MACRO FOR GENERALIZING ANL INSTRUCTION				
-		= 440	MANL MACRO DEST, SRC				
-		= 441	?BINOP ANL, DEST, SRC	00000000 00000000 00000000 00000000	00000000		
-		= 442	ENDM				
-		= 443	;				
-		= 444	MORL MACRO FOR GENERALIZING ORL INSTRUCTION				
-		= 445	MORL MACRO DEST, SRC				
-		= 446	?BINOP ORL, DEST, SRC	00000000 00000000 00000000 00000000	00000000		
-		= 447	ENDM				
-		= 448	;				
-		= 449	MXRL MACRO FOR GENERALIZING XRL INSTRUCTION				
-		= 450	MXRL MACRO DEST, SRC				
-		= 451	?BINOP XRL, DEST, SRC	00000000 00000000 00000000 00000000	00000000		
-		= 452	ENDM				
-		= 453	;				
-		= 454	MXCH MACRO FOR GENERALIZING XCH INSTRUCTION				

LOC	OBJ	LINE	SOURCE STATEMENT	DISPATCH CODE	TIME	LOC	OBJ
-		= 455	MXCH MACRO DEST, SRC	002-7231 000000	0000		
-		= 456	?BINOP XCH, DEST, SRC	002-7231 000000	0000		
-		= 457	ENDM	002-7231 000000	0000		
-		= 458 ;		002-7231 000000	0000		
-		= 459	?UNARY MACRO OPCODE, DEST	002-7231 000000	0000		
-		= 460	?FORM1 MOV, DEST	002-7231 000000	0000		
-		= 461	\$SAVE GEN	002-7231 000000	0000		
-		= 462	OPCODE A	002-7231 000000	0000		
-		= 463	\$RESTORE	002-7231 000000	0000		
-		= 464	?FORM3 DEST	002-7231 000000	0000		
-		= 465	ENDM	002-7231 000000	0000		
-		= 466 ;		002-7231 000000	0000		
-		= 467	MINC MACRO DEST	002-7231 000000	0000		
-		= 468	?UNARY INC, DEST	002-7231 000000	0000		
-		= 469	ENDM	002-7231 000000	0000		
-		= 470 ;		002-7231 000000	0000		
-		= 471	MDEC MACRO DEST	002-7231 000000	0000		
-		= 472	?UNARY DEC, DEST	002-7231 000000	0000		
-		= 473	ENDM	002-7231 000000	0000		
-		= 474 ;		002-7231 000000	0000		
-		= 475	MDJNZ MACRO DEST, ADDR	002-7231 000000	0000		
-		= 476	?UNARY DEC, DEST	002-7231 000000	0000		
-		= 477	\$SAVE GEN	002-7231 000000	0000		
-		= 478	JNZ ADDR	002-7231 000000	0000		
-		= 479	\$RESTORE	002-7231 000000	0000		
-		= 480	ENDM	002-7231 000000	0000		
-		= 481 ;		002-7231 000000	0000		
-		= 482	MRL MACRO DEST	002-7231 000000	0000		
-		= 483	?UNARY RL, DEST	002-7231 000000	0000		
-		= 484	ENDM	002-7231 000000	0000		
-		= 485 ;		002-7231 000000	0000		
-		= 486	MRR MACRO DEST	002-7231 000000	0000		
-		= 487	?UNARY RR, DEST	002-7231 000000	0000		
-		= 488	ENDM	002-7231 000000	0000		
-		= 489 ;		002-7231 000000	0000		
-		= 490	MRRC MACRO DEST	002-7231 000000	0000		
-		= 491	?UNARY RRC, DEST	002-7231 000000	0000		
-		= 492	ENDM	002-7231 000000	0000		
-		= 493 ;		002-7231 000000	0000		
-		= 494	MRLC MACRO DEST	002-7231 000000	0000		
-		= 495	?UNARY RLC, DEST	002-7231 000000	0000		
-		= 496	ENDM	002-7231 000000	0000		
-		= 497 ;		002-7231 000000	0000		
-		= 498	\$EJECT	002-7231 000000	0000		

LOC	OBJ	LINE	SOURCE STATEMENT	THIRDATE 300002	SH11	LOC
		499 ;				
		500 ;=====				
		501 ;=====				
		502 ; BEGINNING OF PROGRAM PROPER				
		503 ;=====				
		504 ;=====				
		505 ;				
		506 ;				
		507 ;=====				
		508 ;				
		509 ; ALLOCATION OF MP I/O PORTS:				
		510 ;				
		511 ;=====				
		512 ;				
		513 ; BUS ;USED FOR BIDIRECTIONAL ADDRESS AND DATA TRANSFERS				
		514 ; P1 ;USED AS INDIVIDUAL CONTROL OUTPUTS AND BREAK LOGIC				
		515 ; P2 ;HIGH-ORDER ADDRESS AND ADDRESS SPACE SELECTION				
		516 ;				
000E		517 PDIGIT EQU P7 ;USED TO ENABLE CHARACTERS AND STROBE ROWS OF KEYBOARD				
000D		518 PSECHI EQU P6 ;USED TO TURN ON HI SEGMENTS OF CURRENTLY ENABLED DIGIT				
000C		519 PSEGLO EQU P5 ;PORT FOR LOWER FOUR SEGMENTS				
000B		520 PINPUT EQU P4 ;PORT USED TO SCAN FOR KEY CLOSURES				
		521 ;				
		522 ;=====				
		523 ;				
		524 ; INDIVIDUAL PINS OF PORT 1 USED AS FOLLOWS:				
		525 ;				
		526 ;=====				
		527 ;				
0001		528 ENDRAM EQU 0000001B ;P10 - HI ENABLES BREAK ON BREAK RAM OUTPUT SIGNAL				
0002		529 ENBLNK EQU 00000010B ;P11 - HI ENABLES BREAK ON RD OR WR TO LINK BY EP				
		530 ; (NOTE: P11 & P10 BOTH HI ENABLES				
		531 ; BREAK ON ANY EP INSTRUCTION CYCLE)				
0004		532 EPSSTP EQU 00000100B ;P12 - LO FORCES EP SS INPUT LOW,				
		533 ; HI GATES BREAKPOINT FLIP-FLOP TO EP SS INPUT.				
0008		534 CLRBFF EQU 00001000B ;P13 - LO CLEARS BREAK FLIP-FLOP				
		535 ; AND ENABLES WR CONTROL TO BREAKPOINT RAM.				
0010		536 EPRSET EQU 00010000B ;P14 - HI RESETS EP				
0020		537 MODOUT EQU 00100000B ;P15 - LO WHEN EP IS EXECUTING USER PROGRAM,				
		538 ; HI WHEN EP FROZEN OR RUNNING OVERLAYS.				
0040		539 TTYOUT EQU 01000000B ;P16 - SERIAL OUTPUT TO TTY OR CRT				
		540 ;P17 - UNUSED				
		541 ;				
		542 \$EJECT				

```

544 ;
545 ; INDIVIDUAL PINS OF PORT 2 USED AS FOLLOWS:
546 ;
547 ; *****
548 ;
549 ; P23-P20 ;ADR11-ADR8 FOR ACCESSING PROGRAM OR DATA RAM ARRAY
550 ;
0010 551 M0 EQU 00010000B ;P24 -- MEMORY MATRIX CONTROL PIN 0
0020 552 M1 EQU 00100000B ;P25 - MEMORY MATRIX CONTROL PIN 1
0040 553 MPUSEL EQU 01000000B ;P26 - HIGH WHEN MP IN CONTROL OF COMMON MEM ARRAY,
554 ; LOW WHEN EP IN CONTROL.
0080 555 EXPMON EQU 10000000B ;P27 -- JUMPERED TO GROUND FOR STANDARD MONITOR,
556 ; FLOATING WHEN EXPANSION MONITOR PRESENT.
557 ;
558 ;
559 ; WHEN MP IN CONTROL OF MEMORY MATRIX M1-M0 USED AS FOLLOWS:
560 ;
561 ; M1 M0 MODE
562 ; 0 0 PROGRAM RAM ARRAY ENABLED FOR READ & WRITE
563 ; 0 1 DATA RAM ARRAY ENABLED FOR READ & WRITE
564 ; 1 X LINK REGISTER ENABLED FOR READ, RAM ARRAYS DISABLED.
565 ; (NOTE: LINK REGISTER ALWAYS ENABLED FOR MP WRITES)
566 ;
567 ; WHEN EP IN CONTROL OF MATRIX M1-M0 USED AS FOLLOWS:
568 ;
569 ; M1 M0 MODE
570 ; 0 X EP PSEN FETCHES FROM LINK REGISTER (USED TO FORCE OPCODES)
571 ; 1 0 EP PSEN FETCHES FROM PROGRAM RAM ARRAY,
572 ; EP RD & WR CONTROL DATA RAM ARRAY.
573 ; 1 1 EP PSEN FETCHES FROM PROGRAM RAM ARRAY,
574 ; EP RD & WR CONTROL LINK REGISTER.
575 ;
576 #EJECT

```


LOC	OBJ	LINE	SOURCE STATEMENT
		577 ;	
		578 ;*****	
		579 ;	
		580 ; SYSTEM CONSTANT DEFINITIONS:	
		581 ;	
		582 ;*****	
		583 ;	
0008		584 DECLARE CHARN0,CONST ;NUMBER OF DIGITS IN DISPLAY AND ROWS OF KEYS	
		598 CHARN0 EQU 8	
		599 ;	
0004		600 DECLARE NCOLS,CONST ;LESSER DIMENSION OF KEYBOARD MATRIX	
		614 NCOLS EQU 4	
		615 ;	
0008		616 DECLARE DEBNCE,CONST ;NUMBER OF SUCESSIVE SCANS BEFORE KEY CLOSURE ACCEPTED	
		630 DEBNCE EQU 8	
		631 ;	
0017		632 DECLARE OVSIZ0,CONST ;SIZE OF LARGEST MINI-MONITOR OVERLAY FOR EP	
		646 OVSIZ0 EQU 23	
		647 ;	
0010		648 DECLARE BUFL0N,CONST ;LENGTH OF HEX FORMAT XMIT BUFFER (MAX RECORD LENGTH)	
		662 BUFL0N EQU 16	
		663 ;	
		664 ;*****	
		665 ;	
		666 ; UTILITY CONSTANT DECLARATIONS	
		667 ;	
		668 ;*****	
		669 ;	
0000		670 DECLARE ZERO,CONST	
		684 ZERO EQU 0	
0001		685 DECLARE PLUS1,CONST	
		699 PLUS1 EQU 1	
0003		700 DECLARE PLUS3,CONST	
		714 PLUS3 EQU 3	
		715 DECLARE NEG1,CONST	
FFFF		729 NEG1 EQU -1	
		730 ;	
		731 \$EJECT	

LOC	OBJ	LINE	SOURCE STATEMENT	ASSEMBLY SOURCE	LINE	OBJ	LOC
		732 ;			732		
		733 ;*****			733		
		734 ;			734		
		735 ; BANK 0 REGISTER ALLOCATION:			735		
		736 ;			736		
		737 ;*****			737		
		738 ;			738		
		739 DECLARE LDAT, RB0 ; DATA USED BY LOGICAL ADDRESSING READ/WRITE UTILITIES			739		
0002		752+ LDAT SET R2			752		
		756 DECLARE KEY, RB0 ; HOLDS KEYCODE RETURNED FROM KBD INPUT ROUTINE.			756		
0003		769+ KEY SET R3			769		
		773 DECLARE ITMP, RB0 ; COUNTER USED AS AN INDEX IN PARSER ROUTINE			773		
0004		786+ ITMP SET R4			786		
		790 DECLARE CHKSUM, RB0 ; CHECKSUM OF DATA BYTES TRANSMITTED IN HEX FILE FORMAT			790		
0005		803+ CHKSUM SET R5			803		
		807 DECLARE DSPTMP, RB0 ; TEMPORARY STORAGE FOR DISPLAY PATTERNS IN 'DSPACC'			807		
0006		820+ DSPTMP SET R6			820		
		824 DECLARE XPCODE, RB0 ; EXPANSION MONITOR ROUTINE CODE NUMBER			824		
0007		837+ XPCODE SET R7			837		
		841 ;			841		
		842 ;*****			842		
		843 ;			843		
		844 ; BANK 1 REGISTER ALLOCATION			844		
		845 ;			845		
		846 ;*****			846		
		847 ;			847		
		848 DECLARE ROTPAT, RB1 ; USED TO HOLD INPUT PATTERN BEING ROTATED THROUGH CY			848		
0002		865+ ROTPAT SET R2			865		
		869 DECLARE ROTCNT, RB1 ; COUNTS NUMBER OF BITS ROTATED THROUGH CY			869		
0003		886+ ROTCNT SET R3			886		
		890 DECLARE LASTKY, RB1 ; HOLDS KEY POSITION OF LAST KEY DEPRESSION DETECTED			890		
0004		907+ LASTKY SET R4			907		
		911 DECLARE CURDIG, RB1 ; HOLDS POSITION OF NEXT CHARACTER TO BE DISPLAYED			911		
0005		928+ CURDIG SET R5			928		
		932 DECLARE KEYFLG, RB1 ; FLAG TO DETECT WHEN ALL KEYS ARE RELEASED			932		
0006		949+ KEYFLG SET R6			949		
		953 ; (REGISTER 7 NOT USED FOR PRIMARY MONITOR)			953		
		954 ;			954		
		955 ;*****			955		
		956 \$EJECT			956		

LOC	OBJ	LINE	SOURCE STATEMENT
		957 ;	
		958 ; *****	
		959 ;	
		960 ; DATA RAM ALLOCATION	
		961 ;	
		962 ; *****	
		963 ;	
		964 DECLARE EPACC, RAM ; STORAGE IN MP FOR EP ACCUMULATOR	
0020		969+ EPACC EQU 32	
		973 DECLARE EPPSW, RAM ; STORAGE IN MP FOR EP PROGRAM STATUS WORD	
0021		970+ EPPSW EQU 33	
		982 DECLARE EPTIMR, RAM ; STORAGE IN MP FOR EP TIMER/COUNTER REGISTER	
0022		987+ EPTIMR EQU 34	
		991 DECLARE EPR0, RAM ; STORAGE IN MP FOR EP REGISTER 0 OF BANK 0	
0023		996+ EPR0 EQU 35	
		1000 DECLARE EPPCLO, RAM ; STORAGE IN MP FOR LOW BYTE OF EP PROGRAM COUNTER	
0024		1005+ EPPCLO EQU 36	
		1009 DECLARE EPPCHI, RAM ; STORAGE IN MP FOR HIGH NIBBLE OF EP PROGRAM COUNTER	
0025		1014+ EPPCHI EQU 37	
		1018 DECLARE HBITLO, RAM ; PARAMETER 1 FOR SERIAL LINK DATA RATE GENERATOR	
0026		1023+ HBITLO EQU 38	
		1027 DECLARE HBITHI, RAM ; PARAMETER 2 FOR SERIAL LINK DATA RATE GENERATOR	
0027		1032+ HBITHI EQU 39	
		1036 DECLARE DSPTIM, RAM ; PARAMETER FOR AUTO-STEP AND AUTO-BREAK SEQUENCING RATE	
0028		1041+ DSPTIM EQU 40	
		1045 DECLARE VERSNO, RAM ; MONITOR VERSION NUMBER	
0029		1050+ VERSNO EQU 41	
		1054 DECLARE HREGA, RAM ; (UNUSED)	
002A		1059+ HREGA EQU 42	
		1063 DECLARE HREGD, RAM ; (UNUSED)	
002B		1068+ HREGD EQU 43	
		1072 DECLARE HREGC, RAM ; (UNUSED)	
002C		1077+ HREGC EQU 44	
		1081 DECLARE HREGD, RAM ; (UNUSED)	
002D		1086+ HREGD EQU 45	
		1090 DECLARE HREGC, RAM ; (UNUSED)	
002E		1095+ HREGC EQU 46	
		1099 DECLARE HREGF, RAM ; (UNUSED)	
002F		1104+ HREGF EQU 47	
		1108 DECLARE SMALO, RAM ; PRIMARY COMMAND STARTING MEMORY ADDRESS (LOW BYTE)	
0030		1113+ SMALO EQU 48	
		1117 DECLARE SMAHI, RAM ; PRIMARY COMMAND STARTING MEMORY ADDRESS (HIGH BYTE)	
0031		1122+ SMAHI EQU 49	
		1126 DECLARE EMALO, RAM ; PRIMARY COMMAND ENDING MEMORY ADDRESS (LOW BYTE)	
0032		1131+ EMALO EQU 50	
		1135 DECLARE EMAHI, RAM ; PRIMARY COMMAND ENDING MEMORY ADDRESS (HIGH BYTE)	
0033		1140+ EMAHI EQU 51	
		1144 DECLARE MEMLO, RAM ; THIRD PARSER PARAMETER & HEX RECORD ADDRESS (LOW)	
0034		1149+ MEMLO EQU 52	
		1153 DECLARE MEMHI, RAM ; THIRD PARSER PARAMETER & HEX RECORD ADDRESS (HIGH)	
0035		1158+ MEMHI EQU 53	
		1162 DECLARE BCODE, RAM ; PRIMARY COMMAND NUMBER FROM PARSER TABLES (0-0)	
0036		1167+ BCODE EQU 54	
		1171 DECLARE TYPE, RAM ; PRIMARY COMMAND MODIFIER/OPTION (0-5)	
0037		1176+ TYPE EQU 55	

	1197	DECLARE	OPTION	EQU	57	; INDEX POINTER USED IN SEARCHING PARSER TABLES
0039	1194		OPTION	EQU	57	
	1198	DECLARE	NEXTPL	RAM		; CHARACTER POSITION FOR DISPLAY UTILITIES TO WRITE NEXT
003A	1203		NEXTPL	EQU	58	
	1207	DECLARE	KDBUF	RAM		; POSITION OF KEY DEBOUNCED BY SCANNING SUBROUTINE
003B	1212		KDBUF	EQU	59	
	1216	DECLARE	KEYLOC	RAM		; INCREMENTED AS SUCCESSIVE KEY LOCATIONS SCANNED
003C	1221		KEYLOC	EQU	60	
	1225	DECLARE	NREPTS	RAM		; KEEPS TRACK OF SUCCESSIVE READS OF SAME KEYSTROKE
003D	1230		NREPTS	EQU	61	
	1234	DECLARE	ASAVE	RAM		; HOLDS ACCUMULATOR VALUE DURING SERVICE ROUTINE
003E	1239		ASAVE	EQU	62	
	1243	DECLARE	RDELAY	RAM		; COUNTER DECREMENTED WHEN AUTO-STEP DELAY IN PROGRESS
003F	1248		RDELAY	EQU	63	
	1252	DECLARE	STRTMP	RAM		; INDEX POINTER FOR DISPLAY CHARACTER STRING ACCESSING
0040	1257		STRTMP	EQU	64	
	1261	DECLARE	BUFCNT	RAM		; COUNT OF DATA BYTES IN HEX FORMAT RECORD BUFFER
0041	1266		BUFCNT	EQU	65	
	1270	DECLARE	RECTYP	RAM		; TYPE OF HEX FORMAT RECORD (0 OR 1)
0042	1275		RECTYP	EQU	66	
	1279	DECLARE	B	RAM		; BIT COUNTER FOR ASCII SERIAL I/O UTILITY SUBROUTINES
0043	1284		B	EQU	67	
	1288	DECLARE	REGC	RAM		; CHARACTER BEING SHIFTED DURING SERIAL I/O PROCESS
0044	1293		REGC	EQU	68	
	1297	DECLARE	H	RAM		; COUNTER IN SOFTWARE DELAY DATA RATE GENERATOR
0045	1302		H	EQU	69	
	1306					
	1307	MBLOCK	SEGMAP	CHARNO		; REGISTER ARRAY FOR DISPLAY PATTERNS
0046	1311		SEGMAP	EQU	70	
	1314					
	1315	MBLOCK	OVBUF	OVSZ		; LOW-ORDER USER PROGRAM DURING MINI-MONITOR OVERLAYS
004E	1319		OVBUF	EQU	78	
	1322					
	1323	MBLOCK	HEXBUF	BUFLN		; ALLOCATE BLOCK OF RAM FOR USE AS HEX RECORD BUFFER
0065	1327		HEXBUF	EQU	101	
	1330					
	1331	#EJECT				

LOC	OBJ	LINE	SOURCE STATEMENT
		1332	DATADLK 40
0300		1337+	ORG 768
		1341	INVALS TABLE OF CONSTANTS TO BE LOADED INTO MP INTERNAL RAM VARIABLES
		1342	AS PART OF SYSTEM INITIALIZATION PROCEDURE:
		1343	
		1344	INITIAL VALUE VARIABLE TYPE
		1345	=====
0300 00		1346	INVALS: DB 00H ; ROTPAT RB1
0301 00		1347	DB 00H ; ROTCNT RB1
0302 00		1348	DB 00H ; LASTKY RB1
0303 00		1349	DB CHARNO ; CURDIG RB1
0304 00		1350	DB 00H ; KEYFLG RB1
0305 00		1351	DB 00H ; <REG7> RB1
0306 00		1352	DB 00H ; EPACC RAM
0307 01		1353	DB 01H ; EPPSW RAM
0308 00		1354	DB 00H ; EPTIMR RAM
0309 00		1355	DB 00H ; EPR0 RAM
030A 00		1356	DB 00H ; EPPCLO RAM
030B 00		1357	DB 00H ; EPPCHI RAM
030C 93		1358	DB 93H ; HBITLO RAM
030D 04		1359	DB 04H ; HBITHI RAM
030E 20		1360	DB 20H ; DSPTIM RAM
030F 25		1361	DB 25H ; VERSNO RAM
0310 00		1362	DB 00H ; HREGA RAM
0311 00		1363	DB 00H ; HREGB RAM
0312 00		1364	DB 00H ; HREGC RAM
0313 00		1365	DB 00H ; HREGD RAM
0314 00		1366	DB 00H ; HREGF RAM
0315 00		1367	DB 00H ; HREGF RAM
0316 00		1368	DB 00H ; SMALO RAM
0317 00		1369	DB 00H ; SMAHI RAM
0318 FF		1370	DB 0FFH ; EMAILO RAM
0319 0F		1371	DB 0FH ; EMAHI RAM
031A 00		1372	DB 00H ; MEMLO RAM
031B 00		1373	DB 00H ; MEMHI RAM
031C 00		1374	DB 00H ; BCODE RAM
031D 04		1375	DB 04H ; TYPE RAM
031E 01		1376	DB 01H ; NUMCON RAM
031F 00		1377	DB 00H ; OPTION RAM
0320 00		1378	DB CHARNO ; NEXTPL RAM
0321 FF		1379	DB 0FFH ; KBDDBUF RAM
0322 00		1380	DB 00H ; KEYLOC RAM
0023		1381	NOVALS EQU \$- INVALS
		1382	SIZECHK
0023		1385+	SIZE SET 35
		1386+	
		1387+	*****
		1396	\$EJECT

LOC	OBJ	LINE	SOURCE STATEMENT	INSTR	OPCODE	DATA	DISP	ADDR	LEN
		1397 \$	INCLUDE(:F0:PARSER.MOD)						
		=1398	CODEBLK 45						
0000		=1403+	ORG 0						
		=1407 ;	INIT						
		=1408 ;	AND RAM LOCATIONS TO DEFINED VALUES.						
0000 C5		=1409 INIT:	SEL R00						
0001 BF00		=1410	MOV XPCODE, #0						
0003 74D1		=1411	CALL XPTST						
0005 27		=1412	CLR A						
0006 3D		=1413	MOVD PSEGLO, A						
0007 3E		=1414	MOVD PSEGLI, A						
0008 D81A		=1415	MOV R0, #1AH ; START AT RD1 (REG2) = RAM LOC 1AH						
000A B923		=1416	MOV R1, #LOW NOVALS						
000C BA00		=1417	MOV R2, #LOW INVALS						
000E FA		=1418 INITLP:	MOV A, R2						
000F E3		=1419	MOVP3 A, @A						
0010 A0		=1420	MOV @R0, A						
0011 18		=1421	INC R0						
0012 1A		=1422	INC R2						
0013 E90E		=1423	DJNZ R1, INITLP						
0015 55		=1424	STRT T						
0016 744F		=1425	CALL EPERK						
0018 L8BB		=1426	MOV R0, #LOW(OV1BAS+OV5SIZE)						
001A 74GA		=1427	CALL OVLORD						
001C 54E5		=1428	CALL COMFIL						
001E D937		=1429	MOV R1, #TYPE						
0020 11		=1430	INC @R1						
0021 34F2		=1431	CALL INCSMA						
0023 54E5		=1432	CALL COMFIL						
0025 99EF		=1433	ANL PL, #(NOT EPRSE1) ; REMOVE EP RESET SIGNAL						
0027 0429		=1434	JMP MAIN						
		=1435 ;							
		=1436	SIZECHK						
0029		=1439+ SIZE	SET 41						
		=1440+;							
		=1441+;	*****						
		=1450 \$EJECT							

LOC	OBJ	LINE	SOURCE STATEMENT	HEX	OBJ	HEX	LOC	OBJ
=1451 ;								
=1452 ;			KEYBOARD LAYOUT:					
=1453 ;			=====					
=1454 ;								
=1455 ;								
=1456 ;	!	!!	!!	!!	!!	!!	!!	!!
=1457 ;	!	LIST	!!GO/RESET!! GO !!EXAM/CHN!! C !! D !! E !! F !!					
=1458 ;	!	!!	!!	!!	!!	!!	!!	!!
=1459 ;								
=1460 ;								
=1461 ;	!	!!PROG BRK!!PROG MEM!!REGISTER!	!!	!!	!!	!!	!!	!!
=1462 ;	!	UPLOAD	!!	8	!!	9	!!	A
=1463 ;	!	!!AUTO STP!!SING STP!! NO BRK	!!	!!	!!	!!	!!	!!
=1464 ;								
=1465 ;								
=1466 ;	!	!!DATA BRK!!DATA MEM!!	!!	!!	!!	!!	!!	!!
=1467 ;	!	DNLOAD	!!	4	!!	5	!!	6
=1468 ;	!	!!AUTO BRK!!WITH BRK!!	!!	!!	!!	!!	!!	!!
=1469 ;								
=1470 ;								
=1471 ;	!	!!	!!	!!	!!	!!	!!	!!
=1472 ;	!	FILL	!!HARD REG!! NEXT/, !! END/	0	!!	1	!!	2
=1473 ;	!	!!	!!	!!	!!	!!	!!	!!
=1474 ;								
=1475 ;								
=1476	\$JECT							

THE FOLLOWING EQUATES DETERMINES HOW THE PARSER INTERPRETS
VALUES RETURNED BY THE KEYBOARD SCANNING INPUT ROUTINE
WHEN THE VARIOUS KEYS OF THE KEYBOARD ARE PRESSED.

```

=1478 ;
=1479 ;
=1480 ;
=1481 ;
=1482 ;
=1483 ;KEY0 EQU 00H VALUE RETURNED FOR EACH KEY OF KEYBOARD MATRIX
=1484 ;KEY1 EQU 01H BY KEYBOARD SCANNING SUBROUTINE "KBDIN".
=1485 ;KEY2 EQU 02H
=1486 ;KEY3 EQU 03H +---+---+---+---+---+---+---+---+
=1487 ;KEY4 EQU 04H ! 1C ! 1D ! 1E ! 1F ! ! 0C ! 0D ! 0E ! 0F !
=1488 ;KEY5 EQU 05H +---+---+---+---+---+---+---+---+
=1489 ;KEY6 EQU 06H ! 18 ! 19 ! 1A ! 1B ! ! 08 ! 09 ! 0A ! 0B !
=1490 ;KEY7 EQU 07H +---+---+---+---+---+---+---+---+
=1491 ;KEY8 EQU 08H ! 14 ! 15 ! 16 ! 17 ! ! 04 ! 05 ! 06 ! 07 !
=1492 ;KEY9 EQU 09H +---+---+---+---+---+---+---+---+
=1493 ;KEYA EQU 0AH ! 10 ! 11 ! 12 ! 13 ! ! 00 ! 01 ! 02 ! 03 !
=1494 ;KEYB EQU 0BH +---+---+---+---+---+---+---+---+
=1495 ;KEYC EQU 0CH
=1496 ;KEYD EQU 0DH
=1497 ;KEYE EQU 0EH
=1498 ;KEYF EQU 0FH
0010 =1499 KEYFIL EQU 10H ;[FILL COMMAND]
0012 =1500 KEYNXT EQU 12H ;[NEXT/]
0013 =1501 KEYEND EQU 13H ;[END/]
0014 =1502 KEYREL EQU 14H ;[DOWNLOAD COMMAND]
0015 =1503 KEYPAT EQU 15H ;[AUTOBREAK MODIFIER]
0016 =1504 KEYDM EQU 16H ;[DATA MEMORY MODIFIER]
0017 =1505 KEYCLR EQU 17H ;[CLEAR/PREVIOUS]
0018 =1506 KEYREC EQU 18H ;[UPLOAD COMMAND]
0019 =1507 KEYTRA EQU 19H ;[AUTOSTEP MODIFIER]
001A =1508 KEYPM EQU 1AH ;[PROGRAM MEMORY MODIFIER]
001B =1509 KEYREG EQU 1BH ;[REGISTER MEMORY MODIFIER]
001C =1510 KEVLST EQU 1CH ;[FORMATTED DATA OUTPUT COMMAND]
001D =1511 KCORES EQU 1DH ;[GO FROM RESET STATE COMMAND]
001E =1512 KEYGO EQU 1EH ;[GO COMMAND]
001F =1513 KEYMOD EQU 1FH ;[EXAMINE/MODIFY COMMAND]
0008 =1514 KSETB EQU 08H ;[SET BREAKPOINT COMMAND]
000C =1515 KCLRB EQU 0CH ;[CLEAR BREAKPOINT COMMAND]
=1516 ;
=1517 ;
0019 =1518 PERK EQU 19H ;[PROGRAM BREAKPOINT MEMORY MODIFIER]
0015 =1519 DBRK EQU 15H ;[DATA BREAKPOINT MEMORY MODIFIER]
0011 =1520 RINT EQU 11H ;[HARDWARE REGISTER MEMORY MODIFIER]
001B =1521 NOBRK EQU 1BH ;[WITHOUT BREAKPOINTS MODIFIER]
001C =1522 WBRK EQU 16H ;[WITH BREAKPOINTS ENABLED MODIFIER]
001A =1523 SING EQU 1AH ;[SINGLE STEP MODIFIER]
=1524 ;
=1525 $EJECT

```

LOC	OBJ	LINE	SOURCE STATEMENT	INSTR	OPER	DISP	VAL
		=1526	CODEBLK 160	00000000	0000	0000	0000
0029		=1531+	ORG 41	00000000	0000	0000	0000
		=1535 ;	MAIN OUTPUT_MESSAGE(COMMAND_PROMPT)	00000000	0000	0000	0000
		=1536 ;	CALL INPUT_BYTE(KEY)	00000000	0000	0000	0000
		=1537 ;	MAIN2 IF THE KEY=END GO TO MAIN.	00000000	0000	0000	0000
		=1538 ;		00000000	0000	0000	0000
0029	B701	=1539	MAIN: MOV XPCODE, #1	00000000	0000	0000	0000
002B	74D1	=1540	CALL XPTST	00000000	0000	0000	0000
002D	2301	=1541	MOV A, #1	00000000	0000	0000	0000
002F	3400	=1542	CALL OUTUTL	00000000	0000	0000	0000
0031	14EC	=1543	CALL INPKEY	00000000	0000	0000	0000
0033	FB	=1544	MAIN2: MOV A, KEY	00000000	0000	0000	0000
0034	D313	=1545	XRL A, #KEYEND	00000000	0000	0000	0000
0036	C629	=1546	JZ MAIN	00000000	0000	0000	0000
		=1547 ;		00000000	0000	0000	0000
		=1548 ;	FINDOP FIND OUT IF THE KEY PRESSED IS A LEGITIMATE COMMAND INITIATOR:	00000000	0000	0000	0000
		=1549 ;	ITMP:=CTAB	00000000	0000	0000	0000
		=1550 ;	BCODE:=TYPE:=0	00000000	0000	0000	0000
		=1551 ;	WHILE CTAB(ITMP)<0 /CTAB EXHAUSTED/	00000000	0000	0000	0000
		=1552 ;	IF CTAB(ITMP)=KEY GOTO MAINA /COMMAND ENTRY FOUND IN CTAB/	00000000	0000	0000	0000
		=1553 ;	ELSE ITMP:=ITMP+COMMAND_ENTRY_SIZE	00000000	0000	0000	0000
		=1554 ;	BCODE:=BCODE+1	00000000	0000	0000	0000
		=1555 ;	ENDWHILE	00000000	0000	0000	0000
		=1556 ;	GOTO ERROR	00000000	0000	0000	0000
0038	BC23	=1557	MOV ITMP, #CTAB	00000000	0000	0000	0000
		=1558	MMOV BCODE, ZERO	00000000	0000	0000	0000
003A	B936	=1569+	MOV R1, #BCODE	00000000	0000	0000	0000
003C	B100	=1570+	MOV @R1, #ZERO	00000000	0000	0000	0000
		=1574	MMOV TYPE, ZERO	00000000	0000	0000	0000
003E	B937	=1585+	MOV R1, #TYPE	00000000	0000	0000	0000
0040	B100	=1586+	MOV @R1, #ZERO	00000000	0000	0000	0000
0042	FC	=1590	FINDOP: MOV A, ITMP	00000000	0000	0000	0000
0043	E3	=1591	MOV #3	00000000	0000	0000	0000
0044	B2BC	=1592	JB5 MERROR	00000000	0000	0000	0000
0046	D8	=1593	XRL A, KEY	00000000	0000	0000	0000
0047	C652	=1594	JZ MAINA	00000000	0000	0000	0000
0049	FC	=1595	MOV A, ITMP	00000000	0000	0000	0000
004A	0303	=1596	ADD A, #COMSIZ	00000000	0000	0000	0000
004C	AC	=1597	MOV ITMP, A	00000000	0000	0000	0000
004D	B936	=1598	MOV R1, #BCODE	00000000	0000	0000	0000
004F	11	=1599	INC @R1	00000000	0000	0000	0000
0050	0442	=1600	JMP FINDOP	00000000	0000	0000	0000
		=1601 ;		00000000	0000	0000	0000
		=1602 ;	OUTPUT_MESSAGE(STRCOM(BCODE)) /*PROMPT FOR THE CURRENT COMMAND*/	00000000	0000	0000	0000
		=1603 ;	I:=I+1	00000000	0000	0000	0000
		=1604 ;	OPTION:=MEM(I)	00000000	0000	0000	0000
		=1605 ;	I:=I+1	00000000	0000	0000	0000
		=1606 ;	NO_OF_PARAMETERS:=MEM(I)	00000000	0000	0000	0000
		=1607 ;	I:=3	00000000	0000	0000	0000
		=1608 ;		00000000	0000	0000	0000
		=1609	MAINR: MMOV A, BCODE	00000000	0000	0000	0000
0052	B936	=1618+	MOV R1, #BCODE	00000000	0000	0000	0000
0054	F1	=1619+	MOV A, @R1	00000000	0000	0000	0000
0055	031D	=1623	ADD A, #STRCOM	00000000	0000	0000	0000
0057	3402	=1624	CALL OUTCLR	00000000	0000	0000	0000

LOC	OBJ	LINE	SOURCE STATEMENT	INSTR	OPCODE	DISP	ADDR	DATA
0059	1C	=1625	INC ITMP	0000	0000	0000	0000	0000
005A	FC	=1626	MOV A, ITMP	0000	0000	0000	0000	0000
005B	E3	=1627	MOVP3 A, @A ; GET OPTION POINTER	0000	0000	0000	0000	0000
		=1628	MMOV OPTION, A	0000	0000	0000	0000	0000
005C	B939	=1641+	MOV R1, #OPTION	0000	0000	0000	0000	0000
005E	A1	=1642+	MOV @R1, A	0000	0000	0000	0000	0000
005F	1C	=1646	INC ITMP	0000	0000	0000	0000	0000
0060	FC	=1647	MOV A, ITMP	0000	0000	0000	0000	0000
0061	E3	=1648	MOVP3 A, @A ; GET NO OF PARAMETERS	0000	0000	0000	0000	0000
		=1649	MMOV NUMCON, A	0000	0000	0000	0000	0000
0062	B938	=1662+	MOV R1, #NUMCON	0000	0000	0000	0000	0000
0064	A1	=1663+	MOV @R1, A	0000	0000	0000	0000	0000
		=1667 ;						
		=1668 ;	PARAMETER_BUFFER(0=>5):=0					
		=1669 ;						
0065	B906	=1670	MOV R1, #6 ; EACH PARAM IS 2 BYTES	0000	0000	0000	0000	0000
0067	B030	=1671	MOV R0, #SMALO ; START OF PARAM BUFFERS	0000	0000	0000	0000	0000
0069	B000	=1672 MAINB:	MOV @R0, #00H	0000	0000	0000	0000	0000
006B	18	=1673	INC R0	0000	0000	0000	0000	0000
006C	E969	=1674	DJNZ R1, MAINB	0000	0000	0000	0000	0000
006E	14EC	=1675	CALL INPKEY	0000	0000	0000	0000	0000
		=1676 ;						
		=1677 ;	WHILE KEY>MEM(OPTION+TYPE)[6-0] DO					
		=1678 ;	IF MEM(OPTION+TYPE)[7]=1 GOTO MAIND1					
		=1679 ;	TYPE:=TYPE+1					
		=1680 ;	ENDWHILE					
		=1681 ;						
		=1682	MMOV ITMP, OPTION	0000	0000	0000	0000	0000
0070	B939	=1690+	MOV R1, #OPTION	0000	0000	0000	0000	0000
0072	F1	=1699+	MOV A, @R1	0000	0000	0000	0000	0000
0073	0C	=1712+	MOV ITMP, A	0000	0000	0000	0000	0000
0074	1C	=1715	INC ITMP	0000	0000	0000	0000	0000
		=1716 MAINC1:	MMOV A, ITMP	0000	0000	0000	0000	0000
0075	FC	=1732+	MOV A, ITMP	0000	0000	0000	0000	0000
0076	E3	=1736	MOVP3 A, @A	0000	0000	0000	0000	0000
0077	97	=1737	CLR C	0000	0000	0000	0000	0000
0078	F7	=1738	RLC A	0000	0000	0000	0000	0000
0079	77	=1739	RR A ; STRIP BIT SEVEN INTO CARRY	0000	0000	0000	0000	0000
007A	DB	=1740	XRL A, KEY	0000	0000	0000	0000	0000
007B	C693	=1741	JZ MAIND	0000	0000	0000	0000	0000
007D	F687	=1742	JC MAIND1	0000	0000	0000	0000	0000
		=1743	MINC TYPE	0000	0000	0000	0000	0000
007F	B937	=1748+	MOV R1, #TYPE	0000	0000	0000	0000	0000
0081	F1	=1749+	MOV A, @R1	0000	0000	0000	0000	0000
0082	17	=1753+	INC A	0000	0000	0000	0000	0000
0083	A1	=1758+	MOV @R1, A	0000	0000	0000	0000	0000
0084	1C	=1761	INC ITMP	0000	0000	0000	0000	0000
0085	0475	=1762	JMP MAINC1	0000	0000	0000	0000	0000
		=1763 ;						
		=1764 ;	MODIFIER NOT FOUND SO RESET TYPE INDEX TO DEFAULT CASE (ZERO).					
		=1765 ;						
		=1766 MAIND1:	MMOV TYPE, ZERO	0000	0000	0000	0000	0000
0087	B937	=1777+	MOV R1, #TYPE	0000	0000	0000	0000	0000
0089	B100	=1778+	MOV @R1, #ZERO	0000	0000	0000	0000	0000
		=1782	MMOV A, OPTION	0000	0000	0000	0000	0000

LOC	OBJ	LINE	SOURCE STATEMENT	OBJECT STATEMENT	LOC	OBJ
008B	B939	=1791+	MOV RL,#OPTION	MOV RL,#OPTION	008B	B939
008D	F1	=1792+	MOV A,@R1	MOV A,@R1	008D	F1
008E	E3	=1796	MOVFP3 A,@A	MOVFP3 A,@A	008E	E3
008F	3404	=1797	CALL OUTMSG	CALL OUTMSG	008F	3404
0091	049E	=1798	JMP MAINB0	JMP MAINB0	0091	049E
		=1799 ;				
		=1800 ;	CALL OUTPUT_MESSAGE(MODIFIER)	CALL OUTPUT_MESSAGE(MODIFIER)		
		=1801 MAIND:	MMOV A,OPTION	MMOV A,OPTION		
0093	B939	=1810+	MOV RL,#OPTION	MOV RL,#OPTION	0093	B939
0095	F1	=1811+	MOV A,@R1	MOV A,@R1	0095	F1
0096	E3	=1815	MOVFP3 A,@A	MOVFP3 A,@A	0096	E3
		=1816	MADD A,TYPE	MADD A,TYPE		
0097	B937	=1822+	MOV RL,#TYPE	MOV RL,#TYPE	0097	B937
0099	61	=1823+	ADD A,@R1	ADD A,@R1	0099	61
009A	3404	=1827	CALL OUTMSG	CALL OUTMSG	009A	3404
009C	14EC	=1828	CALL INPKEY	CALL INPKEY	009C	14EC
		=1829 ;				
009E	BC00	=1830 MAINB0:	MOV ITMP,#0	MOV ITMP,#0	009E	BC00
00A0	2330	=1831 MAINB1:	MOV A,#SMALO	MOV A,#SMALO	00A0	2330
00A2	6C	=1832	ADD A,ITMP	ADD A,ITMP	00A2	6C
00A3	6C	=1833	ADD A,ITMP	ADD A,ITMP	00A3	6C
00A4	A8	=1834	MOV R0,A	MOV R0,A	00A4	A8
00A5	14C0	=1835	CALL INPADR	CALL INPADR	00A5	14C0
00A7	FCBA	=1836	JC CMDINT	JC CMDINT	00A7	FCBA
00A9	1C	=1837	INC ITMP	INC ITMP	00A9	1C
00AA	B938	=1838	MOV RL,#NUMCON	MOV RL,#NUMCON	00AA	B938
00AC	F1	=1839	MOV A,@R1	MOV A,@R1	00AC	F1
00AD	07	=1840	DEC A	DEC A	00AD	07
00AE	A1	=1841	MOV @R1,A	MOV @R1,A	00AE	A1
00AF	C6BA	=1842	JZ CMDINT	JZ CMDINT	00AF	C6BA
00B1	FB	=1843	MOV A,KEY	MOV A,KEY	00B1	FB
00B2	D313	=1844	XRL A,#KEYEND	XRL A,#KEYEND	00B2	D313
00B4	C6BA	=1845	JZ CMDINT	JZ CMDINT	00B4	C6BA
00B6	14EC	=1846	CALL INPKEY	CALL INPKEY	00B6	14EC
00B8	04A0	=1847	JMP MAINB1	JMP MAINB1	00B8	04A0
		=1848 ;				
		=1849 ;	CMDINT ENTER THE COMMAND PROCESSOR WITH:			
		=1850 ;	BASE_CODE=THE MAIN COMMAND TYPE			
		=1851 ;	TYPE=SUBCOMMAND TYPE			
		=1852 ;	PARAMETER(1)=FIRST ADDRESS			
		=1853 ;	PARAMETER(2)=SECOND ADDRESS			
		=1854 ;	PARAMETER(3)=DATA			
00BA	4400	=1855 CMDINT:	JMP IMPLM	JMP IMPLM	00BA	4400
		=1856 ;				
		=1857 ;	MERRR ERROR ENCOUNTERED IN MAIN PARSING ROUTINE.			
00BC	BA01	=1858 MERRR:	MOV LDAT,#1	MOV LDAT,#1	00BC	BA01
00BE	249A	=1859	JMP PERRR	JMP PERRR	00BE	249A
		=1860	SIZECHK			
0097		=1863+	SIZE SET 151			
		=1864+;				
		=1865+;*****				
		=1874 \$EJECT				

```

0525      =1880+      ORG      803
      =1884 ;
      =1885 ; *****
      =1886 ;
      =1887 ;      TABLES FOR PARSER
      =1888 ;
      =1889 ; *****
      =1890 ;
      =1891 ;      THE CTAB TABLE CONTAINS <COMSIZ> ENTRIES FOR EACH COMMAND. THE MEANING
      =1892 ;      OF THE ENTRIES IS AS FOLLOWS:
      =1893 ;
      =1894 ;      ENTRY 0. COMMAND KEY TO INITIATE
      =1895 ;      ENTRY 1. POINTER TO THE LIST OF OPTIONS APPLICABLE TO THIS COMMAND
      =1896 ;      ENTRY 2. NUMBER OF NUMERIC PARAMETERS REQUIRED BY THE COMMAND
      =1897 ;
0023      =1898 CTAB EQU      $ AND 0FFH
0003      =1899 COMSIZ EQU      3
      =1900 ;
0323 1F      =1901 DB      KEYMOD, LOW OPTAB1, 1      ; EXAM
0324 3F      =
0325 01      =
0326 1E      =1902 DB      KEYGO, LOW OPTAB3, 1      ; GO
0327 49      =
0328 01      =
0329 10      =1903 DB      KEYFIL, LOW OPTAB1, 3      ; FILL
032A 3F      =
032B 03      =
032C 1C      =1904 DB      KEYLST, LOW OPTAB1, 2      ; DUMP
032D 3F      =
032E 02      =
032F 18      =1905 DB      KEYREC, LOW OPTAB1, 2      ; RECORD
0330 3F      =
0331 02      =
0332 14      =1906 DB      KEYREL, LOW OPTAB1, 0      ; RELOAD
0333 3F      =
0334 00      =
0335 00      =1907 DB      KSE1B, LOW OPTAB2, 1      ; SETBRK
0336 46      =
0337 01      =
0338 0C      =1908 DB      KCLRB, LOW OPTAB2, 1      ; CLRBRK
0339 46      =
033A 01      =
033B 1D      =1909 DB      KGORES, LOW OPTAB3, 0      ; GO FROM RESET STATE
033C 49      =
033D 00      =
033E FF      =1910 DB      0FFH      ; ESCOP
      =1911 ;
      =1912 $EJECT

```


LOC	OBJ	LINE	SOURCE STATEMENT	THIRGHTZ EXRGE2	LOC	LOC
		=1913 ;				
		=1914 ;	THE OPTION TABLE GIVES THE VARIOUS OPTIONS ALLOWED FOR EACH			
		=1915 ;	BASIC COMMAND, AS FOLLOWS:			
		=1916 ;				
		=1917 ;	ENTRY 0. START OF TABLE OF MODIFIER RESPONSES.			
		=1918 ;	ENTRY 1+. ALLOWED MODIFIER KEYSTROKES CORRESPONDING TO OPTIONS 0-5.			
		=1919 ;	NOTE THAT THE LAST BYTE IN EACH OPTION GROUP HAS BIT			
		=1920 ;	SEVEN SET TO INDICATE THE END.			
		=1921 ;				
033F	26	=1922 OPTAB1: DB	STRMEM			
0340	1A	=1923 DB	KEYPM, KEYDM, KEYREG, RINT			
0341	16	=				
0342	18	=				
0343	11	=				
0344	19	=1924 DB	PBRK, DBRK OR 80H			
0345	95	=				
0346	26	=1925 OPTAB2: DB	STRMEM			
0347	1A	=1926 DB	KEYPM, KEYDM OR 80H			
0348	96	=				
0349	2C	=1927 OPTAB3: DB	STRGOC			
034A	1B	=1928 DB	NOBRK, WBRK, SING			
034B	16	=				
034C	1A	=				
034D	15	=1929 DB	KEYPAT, KEYTRA OR 80H			
034E	99	=				
		=1930	SIZECHK			
002C		=1933+ SIZE SET 44				
		=1934+;				
		=1935+; *****				
		=1944 \$EJECT				

LOC	OBJ	LINE	SOURCE STATEMENT	DISPATCH SYMBOL	VAL	LOC	OBJ
		=1945	CODEBLK 130				
0100		=1955+	ORG 256				
		=1959	OUTUTL OUTPUT ONE OF FOUR UTILITY DISPLAY PROMPTS (LEFT JUSTIFIED)				
		=1960	ACCORDING TO ACC CONTENTS (0-3).				
		=1961	OUTCLR CLEAR DISPLAY AND OUTPUT CHARACTER STRING STARTING				
		=1962	AT THE ADDRESS POINTED TO BY BYTE AT ADDRESS IN ACCUMULATOR.				
		=1963	OUTMSG SUBROUTINE TO COPY A STRING OF BIT PATTERNS FROM ROM TO THE				
		=1964	DISPLAY REGISTERS.				
		=1965	STRING SELECTED IS DETERMINED BY ACC WHEN CALLED.				
		=1966	ON ENTERING OUTMSG, ACC CONTENTS ARE USED TO ADDRESS A BYTE IN A				
		=1967	LOOKUP TABLE ON THE CURRENT PAGE WHICH CONTAINS THE ADDRESS OF				
		=1968	A STRING OF SEGMENT PATTERN DATA BYTES TO BE PRINTED UNTO THE				
		=1969	DISPLAY.				
		=1970	THE END OF THE STRING IS INDICATED WHEN BIT7 =1				
		=1971	CALLS SUBROUTINE 'WDISP'				
		=1972	TO ACTUALLY EFFECT WRITING INTO THE DISPLAY REGISTERS.				
0100	0319	=1973	OUTUTL: ADD A, #STRUTL				
0102	B4F1	=1974	OUTCLR: CALL CLEAR				
0104	A3	=1975	OUTMSG: MOVP A, @A				
		=1976	MMOV STRTMP, A				
0105	B940	=1980+	MOV R1, #STRTMP				
0107	A1	=1990+	MOV @R1, A				
		=1994	PRNT2: MMOV A, STRTMP ; LOAD NEXT CHARACTER LOCATION				
0108	B940	=2003+	MOV R1, #STRTMP				
010A	F1	=2004+	MOV A, @R1				
010B	A3	=2008	MOVP A, @A ; LOAD BIT PATTERN INDIRECT				
010C	F217	=2009	JB7 PRNT1				
010E	D4D8	=2010	CALL WDISP ; OUTPUT TO NEXT CHARACTER POSITION				
		=2011	MINC STRTMP ; INDEX POINTER				
0110	B940	=2016+	MOV R1, #STRTMP				
0112	F1	=2017+	MOV A, @R1				
0113	17	=2021+	INC A				
0114	A1	=2026+	MOV @R1, A				
0115	2408	=2029	JMP PRNT2				
0117	C4D8	=2030	PRNT1: JMP WDISP ; DONE				
		=2031					
0019		=2032	STRUTL EQU LOW \$				
0119	31	=2033	DB LOW(DERROR) ; UTILITY MESSAGE 0 ADDRESS				
011A	37	=2034	DB LOW(DSGNON) ; UTILITY MESSAGE 1 ADDRESS				
011B	3E	=2035	DB LOW(DRUN) ; UTILITY MESSAGE 2 ADDRESS				
011C	44	=2036	DB LOW(DBPNT) ; UTILITY MESSAGE 3 ADDRESS				
001D		=2037	STRCOM EQU LOW \$				
011D	46	=2038	DB LOW(DMOD) ; BASIC COMMAND 0 RESPONSE ADDRESS				
011E	49	=2039	DB LOW(DGO) ; BASIC COMMAND 1 RESPONSE ADDRESS				
011F	4B	=2040	DB LOW(DFILL) ; BASIC COMMAND 2 RESPONSE ADDRESS				
0120	4E	=2041	DB LOW(DLST) ; BASIC COMMAND 3 RESPONSE ADDRESS				
0121	51	=2042	DB LOW(DREC) ; BASIC COMMAND 4 RESPONSE ADDRESS				
0122	54	=2043	DB LOW(DREL) ; BASIC COMMAND 5 RESPONSE ADDRESS				
0123	57	=2044	DB LOW(DSB) ; BASIC COMMAND 6 RESPONSE ADDRESS				
0124	5A	=2045	DB LOW(DCB) ; BASIC COMMAND 7 RESPONSE ADDRESS				
0125	5D	=2046	DB LOW(DGR) ; BASIC COMMAND 8 RESPONSE ADDRESS				
0026		=2047	STRMEM EQU LOW \$				
0126	5F	=2048	DB LOW(DPRMEM) ; DATA TYPE MODIFIER 0 RESPONSE ADDRESS				
0127	61	=2049	DB LOW(DDMEM) ; DATA TYPE MODIFIER 1 RESPONSE ADDRESS				
0128	63	=2050	DB LOW(DRM) ; DATA TYPE MODIFIER 2 RESPONSE ADDRESS				

LOC	OBJ	LINE	SOURCE STATEMENT	DISPATCH	TIME	DATE
0129	69	=2051	DB LOW(DINTRG) ; DATA TYPE MODIFIER 3 RESPONSE ADDRESS			
012A	65	=2052	DB LOW(DPRBRK) ; DATA TYPE MODIFIER 4 RESPONSE ADDRESS			
012B	67	=2053	DB LOW(DDABRK) ; DATA TYPE MODIFIER 5 RESPONSE ADDRESS			
002C		=2054	STRGOC EQU LOW \$			
012C	68	=2055	DB LOW(DNDBRK) ; EXECUTION MODE MODIFIER 0			
012D	60	=2056	DB LOW(DWBRK) ; EXECUTION MODE MODIFIER 1			
012E	6F	=2057	DB LOW(DSS) ; EXECUTION MODE MODIFIER 2			
012F	72	=2058	DB LOW(DPA) ; EXECUTION MODE MODIFIER 3			
0130	75	=2059	DB LOW(DTR) ; EXECUTION MODE MODIFIER 4			
		=2060 ;				
		=2061 ;	UTILITY OUTPUT MESSAGES			
		=2062 ;				
		=2063	ERROR:			
0131	79	=2064	DB 01111001B ; "E"			
0132	50	=2065	DB 01010000B ; "R"			
0133	50	=2066	DB 01010000B ; "R"			
0134	5C	=2067	DB 01011100B ; "O"			
0135	50	=2068	DB 01010000B ; "R"			
0136	C0	=2069	DB 11000000B ; "-."			
		=2070	DSGNON:			
0137	00	=2071	DB 00000000B ; " "			
0138	76	=2072	DB 01110110B ; "H"			
0139	6D	=2073	DB 01101101B ; "S"			
013A	79	=2074	DB 01111001B ; "E"			
013B	40	=2075	DB 01000000B ; "-"			
013C	66	=2076	DB 01100110B ; "4"			
013D	E7	=2077	DB 11100111B ; "9." (TM)			
		=2078	DRUN:			
013E	00	=2079	DB 00000000B ; " "			
013F	40	=2080	DB 01000000B ; "-"			
0140	50	=2081	DB 01010000B ; "R"			
0141	1C	=2082	DB 00011100B ; "U"			
0142	54	=2083	DB 01010100B ; "N"			
0143	C0	=2084	DB 11000000B ; "-."			
		=2085	DEFNT:			
0144	73	=2086	DB 01110011B ; "P"			
0145	B9	=2087	DB 10111001B ; "C."			
		=2088	\$EJECT			

```

-2090 ;
=2090 ; PRIMARY COMMAND RESPONSE STRING PATTERNS
=2091 ;
=2092 DMOD:
0146 79      =2093      DB      01111001B, 00111001B, 11110100B ; "ECH. "
0147 39      =
0148 F4      =
=2094 DGO:
0149 3D      =2095      DB      00111101B, 11011100B ; "GO. "
014A DC      =
=2096 DFILL:
014B 71      =2097      DB      01110001B, 00110000B, 10111000B ; "FIL. "
014C 30      =
014D B8      =
=2098 DLST:
014E 38      =2099      DB      00111000B, 01101101B, 11111000B ; "LST. "
014F 6D      =
0150 F8      =
=2100 DREC:
0151 3E      =2101      DB      00111110B, 01110011B, 10111000B ; "UPL. "
0152 73      =
0153 B8      =
=2102 DREL:
0154 5E      =2103      DB      01011110B, 01010100B, 10111000B ; "DNL. "
0155 54      =
0156 B8      =
=2104 DSB:
0157 6D      =2105      DB      01101101B, 01111000B, 11111100B ; "STB. "
0158 78      =
0159 FC      =
=2106 DCD:
015A 39      =2107      DB      00111001B, 00111000B, 11111100B ; "CLB. "
015B 38      =
015C FC      =
=2108 DGR:
015D 3D      =2109      DB      00111101B, 11010000B ; "GR. "
015E D0      =
=2110 $EJECT

```

LOC	OBJ	LINE	SOURCE STATEMENT
		=2111 ;	
		=2112 ;	MEMORY SPACE MODIFIER OPTION RESPONSE STRINGS:
		=2113 ;	
		=2114 DFRMEM:	
015F	73	=2115	DB 01110011B, 11010000B ; "PR. "
0160	D0	=	
		=2116 DDAMEM:	
0161	5E	=2117	DB 01011110B, 11110111B ; "DA. "
0162	F7	=	
		=2118 DRM:	
0163	50	=2119	DB 01010000B, 10111101B ; "RG. "
0164	BD	=	
		=2120 DFRBKK:	
0165	73	=2121	DB 01110011B, 11111100B ; "PB. "
0166	FC	=	
		=2122 DDABRK:	
0167	5E	=2123	DB 01011110B, 11111100B ; "DB. "
0168	FC	=	
		=2124 DINTRG:	
0169	76	=2125	DB 01110110B, 11010000B ; "HR. "
016A	D0	=	
		=2126 ;	
		=2127 ;	RESPONSE MESSAGES FOR GO CONDITION MODIFIERS.
		=2128 ;	
		=2129 DNOBRK:	
016B	54	=2130	DB 01010100B, 11111100B ; "NB. "
016C	FC	=	
		=2131 DMBRK:	
016D	7C	=2132	DB 01111100B, 11010000B ; "BR. "
016E	D0	=	
		=2133 DSS:	
016F	6D	=2134	DB 01101101B, 01101101B, 11111000B ; "SST. "
0170	6D	=	
0171	F8	=	
		=2135 DPA:	
0172	77	=2136	DB 01110111B, 01111100B, 11010000B ; "ABR. "
0173	7C	=	
0174	D0	=	
		=2137 DTR:	
0175	77	=2138	DB 01110111B, 01101101B, 11111000B ; "AST. "
0176	6D	=	
0177	F8	=	
		=2139 ;	
		=2140	SIZECHK
0078		=2143+	SIZE SET 120
		=2144+;	
		=2145+; *****	
		=2154 \$EJECT	

LOC	OBJ	LINE	SOURCE STATEMENT	SYMBOLS	VALUES	LOC	OBJ
		=2155	CODEBLK 45				
00C0		=2160+	ORG 192				
		=2164 ;	INPADR INPUT DATA INTO TWO-BYTE PARAMETER BUFFER INDICATED BY R0.				
		=2165 ;	RECEIVE NUMERIC KEYS FROM KEYBOARD UNTIL ',' OR '.'.				
		=2166 ;	SHIFT INTO ADDRESS BUFFER;				
		=2167 ;	RE-WRITE DISPLAY.				
		=2168 ;	IF NUMBER OF CONSTANTS NEEDED IS ZERO, NO NEW PARAMETERS ARE ALLOWED.				
		=2169 ;					
00C0 97		=2170 INPADR:	CLR C				
00C1 A7		=2171	CPL C				
		=2172	MMOV A, NUMCON				
00C2 B938		=2181+	MOV R1, #NUMCON				
00C4 F1		=2182+	MOV A, R1				
00C5 C6D7		=2186	JZ ELSIF1				
00C7 FB		=2187 INPAD1:	MOV A, KEY				
00C8 92D7		=2188	JB4 ELSIF1				
00CA 20		=2189	XCH A, R0				
00CB 47		=2190	SWAP A				
00CC 20		=2191	XCH A, R0				
00CD 30		=2192	XCHD A, R0				
00CE 18		=2193	INC R0				
00CF 30		=2194	XCHD A, R0				
00D0 3478		=2195	CALL UPDADR				
00D2 14EC		=2196	CALL INPKEY				
00D4 97		=2197	CLR C				
00D5 04C7		=2198	JMP INPAD1				
		=2199 ;					
		=2200 ;	ELSIF1 IF KEY=', ' OR '.' THEN RETURN.				
		=2201 ;					
00D7 FB		=2202 ELSIF1:	MOV A, KEY				
00D8 D312		=2203	XRL A, #KEYNXT				
00DA C6E5		=2204	JZ ELSIF2				
00DC FB		=2205	MOV A, KEY				
00DD D313		=2206	XRL A, #KEYEND				
00DF C6E5		=2207	JZ ELSIF2				
		=2208 ;					
		=2209 ;	ELSE GOTO PERROR.				
		=2210 ;					
00E1 B802		=2211	MOV LDATA, #2				
00E3 249A		=2212	JMP PERROR				
00E5 B846		=2213 ELSIF2:	MOV R0, #SEGMAP				
00E7 B903		=2214	MOV R1, #3				
00E9 B4F5		=2215	CALL DBLANK				
00EB 83		=2216	RET				
		=2217	SIZECHK				
002C		=2220+ SIZE	SET 44				
		=2221+;					
		=2222+;	*****				
		=2231	\$EJECT				

LOC	OBJ	LINE	SOURCE STATEMENT	INSTR	OPCODE	VAL	ADR	DISP
		=2232	CODEBLK 35					
0178		=2242+	ORG 376					
		=2246 ;	UPADR: UPDATE ADDRESS FIELD					
		=2247 ;	(LAST THREE CHARACTERS OF DISPLAY) WITH ADDRESS BUFFER					
		=2248 UPADR:	MNOV NEXTPL, PLUS3					
0178 B93A		=2259+	MOV R1, #NEXTPL					
017A D103		=2260+	MOV @R1, #PLUS3					
		=2264 ;	WRITE ADDR INTO NEXT THREE BUFFER LOCATIONS.					
017C F0		=2265 UPADR1:	MOV A, @R0					
017D C0		=2266	DEC R0					
017E 530F		=2267	ANL A, #0FH					
0180 960E		=2268	JNZ DSPHI					
0182 D4D8		=2269	CALL WDISP					
0184 F0		=2270	MOV A, @R0					
0185 47		=2271	SWAP A					
0186 530F		=2272	ANL A, #0FH					
0188 9692		=2273	JNZ DSPM1					
018A D4D8		=2274	CALL WDISP					
018C 2494		=2275	JMP DSPLO					
018E D4D3		=2276 DSPHI:	CALL DSPACC					
0190 F0		=2277 DSPMID:	MOV A, @R0					
0191 47		=2278	SWAP A					
0192 D4D3		=2279 DSPM1:	CALL DSPACC					
0194 F0		=2280 DSPLO:	MOV A, @R0					
0195 D4D3		=2281	CALL DSPACC					
0197 83		=2282	RET					
		=2283	SIZECHK					
0020		=2286+ SIZE	SET 32					
		=2287+;						
		=2288+;	*****					
		=2297	\$EJECT					

ADDRESS	HEX	ASSEMBLY	COMMENT
	=2313 ;	OUTPUT_MESSAGE(PERROR_PROMPT)	
	=2314 ;	OUTPUT(LDATA)	
	=2315 ;	CALL INPUT_BYTE(KEY)	
	=2316 ;	UNTIL KEY='CLEAR/PKEYVIOUS'	
0190 BA04	=2317 ERROR: MOV	LDATA, #4	
019A BF02	=2318 ERROR: MOV	XPCODE, #2	
019C 74D1	=2319	CALL XPTST	
019E 27	=2320	CLR A	
019F D7	=2321	MOV PSM, A	
01A0 FB	=2322	MOV A, KEY	
01A1 D317	=2323	XRL A, #KEYCLR	
01A3 C6D6	=2324	JZ ERROR2	
01A5 27	=2325	CLR A	
01A6 3400	=2326	CALL OUTUTL	
01A8 FA	=2327	MOV A, LDATA	
01A9 D4D3	=2328	CALL DSPACC	
	=2329	MMOV KBDBUF, NEG1	
01AB B93B	=2340+	MOV R1, #KDBBUF	
01AD B1FF	=2341+	MOV @R1, #NEG1	
01AF 14EC	=2345	CALL INPKEY	
01B1 FB	=2346	MOV A, KEY	
01B2 D313	=2347	XRL A, #KEYEND	
01B4 9698	=2348	JNZ ERROR	
01B6 0429	=2349 ERROR2: JMP	MAIN	
	=2350	SIZECHK	
0020	=2353+ SIZE SET	32	
	=2354+;		
	=2355+; *****		
	=2364 ;		
	=2365	CODEBLK 00	
0200	=2380+	ORG	512
	=2384 ; IMPLM	IMPLEMENT COMMAND	
0200 2306	=2385 IMPLM: MOV	A, #LOW(JMPTBL)	
	=2386	MADD A, BCODE	
0202 B936	=2392+	MOV R1, #BCODE	
0204 61	=2393+	ADD A, @R1	
0205 B3	=2397	JMPP @A	
	=2398 ;		
	=2399 JMPTBL:		
0206 0F	=2400	DB LOW(JTOMOD)	
0207 20	=2401	DB LOW(JTOGO)	
0208 22	=2402	DB LOW(JTOFIL)	
0209 1A	=2403	DB LOW(JTOLST)	
020A 11	=2404	DB LOW(JTOREC)	
020B 16	=2405	DB LOW(JTOREL)	
020C 2C	=2406	DB LOW(COMSBR)	
020D 28	=2407	DB LOW(COMCBR)	
020E 26	=2408	DB LOW(JGORES)	
	=2409 ;		
020F 444F	=2410 JTOMOD: JMP	EXAMIN	
	=2411 ;		
0211 85	=2412 JTOREC: CLR	F0 ; F0=0 ==> HEX FORMAT DATA DUMP	

LOC	OBJ	LINE	SOURCE STATEMENT	SYMBOL TABLE	VALUE	LOC	OBJ
0212	B472	=2413	CALL HFILED				
0214	0429	=2414	JMP MAIN				
		=2415 ;					
0216	5497	=2416	JTOREL: CALL HRECIN				
0218	0429	=2417	JMP MAIN				
		=2418 ;					
021A	85	=2419	JTOLST: CLR F0				
021B	95	=2420	CPL F0				
021C	B472	=2421	CALL HFILED				
021E	0429	=2422	JMP MAIN				
		=2423 ;					
0220	0400	=2424	JTOGO: JMP EPRUN				
		=2425 ;					
0222	54E5	=2426	JTOFIL: CALL COMFIL				
0224	0429	=2427	JMP MAIN				
		=2428 ;					
0226	8461	=2429	JGORES: JMP COMGOR				
		=2430 ;					
		=2431 ; COMCBR COMMAND TO CLEAR BREAKPOINTS					
0228	B800	=2432	COMCBR: MOV LDATA, #0				
022A	442E	=2433	JMP BRKFIL				
		=2434 ;					
		=2435 ; COMSBR COMMAND TO SET BREAKPOINTS					
022C	B801	=2436	COMSBR: MOV LDATA, #1				
022E	2304	=2437	BRKFIL: MOV A, #4				
		=2438	MADD TYPE, A				
0230	B837	=2448+	MOV R1, #TYPE				
0232	61	=2449+	ADD A, @R1				
0233	R1	=2455+	MOV @R1, A				
0234	F400	=2459	BRKXKT: CALL LSTORE				
0236	FB	=2460	MOV A, KEY				
0237	D313	=2461	XRL A, #KEYEND				
0239	C64D	=2462	JZ BRKEND				
023B	14EC	=2463	CALL INPKEY				
		=2464	MMOV NUMCON, PLUS1				
023D	B938	=2475+	MOV R1, #NUMCON				
023F	D101	=2476+	MOV @R1, #PLUS1				
0241	B830	=2480	MOV R0, #SMALO				
0243	B000	=2481	MOV @R0, #0				
		=2482	MMOV SMAHI, ZERO				
0245	B931	=2493+	MOV R1, #SMAHI				
0247	B100	=2494+	MOV @R1, #ZERO				
0249	14C0	=2498	CALL INPADR				
024B	E634	=2499	JNC BRKXKT				
024D	0429	=2500	BRKEND: JMP MAIN				
		=2501	SIZECHK				
004F		=2504+	SIZE SET 79				
		=2505+;					
		=2506+; *****					
		=2515	\$EJECT				

LOC	OBJ	LINE	SOURCE STATEMENT	INSTR	OP	VAL	DISP	LEN
		=2516	CODEBLK 75	00000000	LD	0000	0000	0000
024F		=2531	ORG 591	00000000	LD	0000	0000	0000
		=2535	; EXAMIN EXAMINE/MODIFY MEMORY COMMAND.					
		=2536	; DISPLAYS MEMORY ADDRESS SPACE OPTION, ADDRESS VALUE, AND CURRENT DATA.					
		=2537	; READS KEYBOARD AND INTERPRETS RESPONSE.					
		=2538						
		=2539	OUTPUT_MESSAGE(<MEMORY_SPACE_OPTION><SMA>,'<DATA_BYTE>')					
024F 85		=2540	EXAMIN: CLR F0	00000000	LD	0000	0000	0000
		=2541	EXAM0: MOV A, TYPE	00000000	LD	0000	0000	0000
0250 0937		=2550	MOV R1, #TYPE	00000000	LD	0000	0000	0000
0252 F1		=2551	MOV A, R1	00000000	LD	0000	0000	0000
0253 0326		=2555	ADD A, #STRMEM ; OFFSET FOR FIRST MEMORY TYPE STRING	00000000	LD	0000	0000	0000
0255 3402		=2556	CALL OUTCLR					
0257 0831		=2557	MOV R0, #SMA0+1	00000000	LD	0000	0000	0000
0259 347C		=2558	CALL UPDAD1					
025B 2348		=2559	MOV A, #01001000B ; '='	00000000	LD	0000	0000	0000
025D 0408		=2560	CALL MDISP					
025F 14FC		=2561	CALL LFETCH					
0261 FA		=2562	MOV A, LDAT	00000000	LD	0000	0000	0000
0262 47		=2563	SWAP A	00000000	LD	0000	0000	0000
0263 0403		=2564	CALL DSPACC					
0265 FA		=2565	MOV A, LDAT	00000000	LD	0000	0000	0000
0266 0403		=2566	CALL DSPACC					
		=2567						
		=2568						
		=2569	INPUT_KEY(KEY)					
		=2570	IF (KEY IS NOT NUMERIC)					
		=2571	IF (KEY=KEYEND) GO TO PARSER					
		=2572	ELSEIF (KEY=KEYNEXT)					
		=2573	INCREMENT <SMA>					
		=2574	GOTO EXAMIN					
		=2575	ELSEIF (KEY=KEYPREVIOUS)					
		=2576	DECREMENT <SMA>					
		=2577	GOTO EXAMIN					
		=2578	ELSE GOTO PERROR					
		=2579						
0260 14EC		=2580	CALL INPKEY					
		=2581	MOV A, KEY	00000000	LD	0000	0000	0000
026A FB		=2597	MOV A, KEY	00000000	LD	0000	0000	0000
026B 927B		=2601	JB4 EXAM1	00000000	LD	0000	0000	0000
		=2602						
		=2603	APPEND DATA WITH <LOWNIB<KEY>>					
		=2604	CALL LSTORE					
		=2605	GOTO EXAMIN					
		=2606						
026D FA		=2607	MOV A, LDAT	00000000	LD	0000	0000	0000
026E 47		=2608	SWAP A					
026F 53F0		=2609	ANL A, #0F0H	00000000	LD	0000	0000	0000
0271 B675		=2610	JF0 EXAM5	00000000	LD	0000	0000	0000
0273 27		=2611	CLR A					
0274 95		=2612	CPL F0					
0275 6B		=2613	EXAM5: ADD A, KEY					
0276 FA		=2614	MOV LDAT, A					
0277 F400		=2615	CALL LSTORE					
0279 4450		=2616	JMP EXAM0					

LOC	OBJ	LINE	SOURCE STATEMENT	ASSEMBLY STATEMENT	LOC	OBJ
		=2617 ;		INCLUDE 'KEYBOARD.MACRO' ;		
027B D313		=2618 EXAM1: XRL A, #(KEYEND)		CODEBLK 236		
027D 9681		=2619 JNZ EXAM2		ORG 0		0000
027F 0429		=2620 JMP MAIN		MAIN EQUATION MODE		
		=2621 ;		HELPER 0 WITH SYSTEM STATUS AND RELEASE		
0281 FB		=2622 EXAM2: MOV A, KEY		DETERMINE IF KEY IS FOLLOWING		
0282 D312		=2623 XRL A, #KEYNXT		IF COMMAND HAS TERMINATED BY THE 'NEXT' KEY		
0284 968A		=2624 JNZ EXAM3		STORE THE DATA TO THE		
0286 34F2		=2625 CALL INCSMA		STORE CP IN THE TOP-OF-STACK (INITIALLY		
0288 444F		=2626 JMP EXAMIN		IF CP IS 0		
028A FB		=2627 EXAM3: MOV A, KEY		IF CP IS 0		
028B D317		=2628 XRL A, #KEYCLR		IF CP IS 1 THEN		
028D 9693		=2629 JNZ EXAM4		IF CP IS ACCUMULATOR		
028F 54F4		=2630 CALL DECSMA				
0291 444F		=2631 JMP EXAMIN				
0293 B703		=2632 EXAM4: MOV LDATA, #03H				
0295 249A		=2633 JMP PERROR				
		=2634 SIZECHK				
0048		=2637+ SIZE SET 72				
		=2638+;				
		=2639+; *****				
		=2648 ;				
		=2649 CODEBLK 4				
00EC		=2654+ ORG 236				
00EC D4C2		=2658 INPKEY: CALL KBDIN ; RETURNS KEY DEPRESSION IN A				
00EE AB		=2659 MOV KEY, A				
00EF 83		=2660 RET				
		=2661 SIZECHK				
0004		=2664+ SIZE SET 4				
		=2665+;				
		=2666+; *****				
		=2675 \$EJECT				

LOC	OBJ	LINE	SOURCE STATEMENT	INSTR	PC	PC+1	PC+2	PC+3	PC+4	PC+5	PC+6	PC+7	PC+8	PC+9	PC+10	PC+11	PC+12	PC+13	PC+14	PC+15	PC+16	PC+17	PC+18	PC+19	PC+20	PC+21	PC+22	PC+23	PC+24	PC+25	PC+26	PC+27	PC+28	PC+29	PC+30	PC+31	
		2676 \$	INCLUDE(:F0:GOCOMS.MOD)																																		
		=2677	CODEBLK 210																																		
0400		=2697+	ORG 1024																																		
		=2701 ;EPRUN	RUN EMULATION MODE.																																		
		=2702 ;	RELOAD EP WITH SYSTEM STATUS AND RELEASE.																																		
		=2703 ;	SEQUENCE IS AS FOLLOWS:																																		
		=2704 ;	IF COMMAND WAS TERMINATED BY THE 'NEXT' KEY:																																		
		=2705 ;	STOKE SMA INTO EP PC;																																		
		=2706 ;	STORE EP PC INTO TOP-OF-STACK (RELATIVE TO EP PSW);																																		
		=2707 ;	PASS EP R0;																																		
		=2708 ;	PASS EP PSW;																																		
		=2709 ;	PASS EP TIMER;																																		
		=2710 ;	PASS EP ACCUMULATOR;																																		
		=2711 ;																																			
0400	2302	=2712 EPRUN:	MOV R, #2																																		
0402	3400	=2713	CALL OUTUTL																																		
		=2714	MMOV R, NUMCON																																		
0404	B938	=2723+	MOV R1, #NUMCON																																		
0406	F1	=2724+	MOV R, @R1																																		
0407	9615	=2728	JNZ EPCONT																																		
		=2729	MMOV EPPCLO, SMAILO																																		
0409	B930	=2745+	MOV R1, #SMAILO																																		
040B	F1	=2746+	MOV R, @R1																																		
040C	B924	=2752+	MOV R1, #EPPCLO																																		
040E	A1	=2753+	MOV @R1, R																																		
		=2756	MMOV EPPCHI, SMAHI																																		
040F	B931	=2772+	MOV R1, #SMAHI																																		
0411	F1	=2773+	MOV R, @R1																																		
0412	B925	=2779+	MOV R1, #EPPCHI																																		
0414	A1	=2780+	MOV @R1, R																																		
0415	FB	=2783	EPCONT: MOV R, KEY																																		
0416	D312	=2784	XRL R, #KEYNXT																																		
0418	C61F	=2785	JZ EPCON1																																		
041A	2301	=2786	MOV R, #01H ; STACK ONE LEVEL DEEP TO HOLD USER STARTING ADDRESS																																		
		=2787	MMOV EPPSW, R																																		
041C	B921	=2800+	MOV R1, #EPPSW																																		
041E	A1	=2801+	MOV @R1, R																																		
		=2805	EPCON1: MMOV LDAT, EPPCLO																																		
041F	B924	=2821+	MOV R1, #EPPCLO																																		
0421	F1	=2822+	MOV R, @R1																																		
0422	AA	=2835+	MOV LDAT, R																																		
		=2838	MMOV R, EPPSW																																		
0423	B921	=2847+	MOV R1, #EPPSW																																		
0425	F1	=2848+	MOV R, @R1																																		
0426	07	=2852	DEC R																																		
0427	5307	=2853	ANL R, #07H																																		
0429	E7	=2854	RL R																																		
042A	0308	=2855	ADD R, #08H																																		
		=2856	MMOV SMAILO, R																																		
042C	B930	=2869+	MOV R1, #SMAILO																																		
042E	A1	=2870+	MOV @R1, R																																		
042F	F4C3	=2874	CALL EPSTOR																																		
		=2875	MINC SMAILO																																		
0431	B930	=2880+	MOV R1, #SMAILO																																		
0433	F1	=2881+	MOV R, @R1																																		

LOC	OBJ	LINE	SOURCE STATEMENT	SYMBOL TABLE	LOC	OBJ
0434	17	=2885+	INC A	(ADDRESS)	00	0000+
0435	R1	=2890+	MOV @R1, A	(ADDRESS)	00	0000+
		=2893	MMOV A, EPPSW	(ADDRESS)	00	0000+
0436	B921	=2902+	MOV R1, #EPPSW	(ADDRESS)	00	0000+
0438	F1	=2903+	MOV A, @R1	(ADDRESS)	00	0000+
0439	53F0	=2907	ANL A, #0F0H	(ADDRESS)	00	0000+
		=2908	MORL A, EPPCHI	(ADDRESS)	00	0000+
043B	B925	=2914+	MOV R1, #EPPCHI	(ADDRESS)	00	0000+
043D	41	=2915+	ORL A, @R1	(ADDRESS)	00	0000+
043E	AR	=2919	MOV LDATH, A	(ADDRESS)	00	0000+
043F	F4C3	=2920	CALL EPSTOR	(ADDRESS)	00	0000+
0441	B8D1	=2921 EPCNT:	MOV R0, #LOW(OV2BAS+OV5IZE)	(ADDRESS)	00	0000+
0443	746A	=2922	CALL OVLORD	(ADDRESS)	00	0000+
		=2923	MMOV A, EPR0	(ADDRESS)	00	0000+
0445	B923	=2932+	MOV R1, #EPR0	(ADDRESS)	00	0000+
0447	F1	=2933+	MOV A, @R1	(ADDRESS)	00	0000+
0448	F4D0	=2937	CALL EPPASS	(ADDRESS)	00	0000+
		=2938	MMOV A, EPPSW	(ADDRESS)	00	0000+
044A	B921	=2947+	MOV R1, #EPPSW	(ADDRESS)	00	0000+
044C	F1	=2948+	MOV A, @R1	(ADDRESS)	00	0000+
044D	F4D0	=2952	CALL EPPASS	(ADDRESS)	00	0000+
		=2953	MMOV A, EPTIMR	(ADDRESS)	00	0000+
044F	B922	=2962+	MOV R1, #EPTIMR	(ADDRESS)	00	0000+
0451	F1	=2963+	MOV A, @R1	(ADDRESS)	00	0000+
0452	F4D0	=2967	CALL EPPASS	(ADDRESS)	00	0000+
		=2968	MMOV A, EPACC	(ADDRESS)	00	0000+
0454	B920	=2977+	MOV R1, #EPACC	(ADDRESS)	00	0000+
0456	F1	=2978+	MOV A, @R1	(ADDRESS)	00	0000+
0457	F4D0	=2982	CALL EPPASS	(ADDRESS)	00	0000+
0459	B903	=2983	ORL P1, #00000011B	(ADDRESS)	00	0000+
045B	F4D0	=2984	CALL EPSTEP	(ADDRESS)	00	0000+
045D	745A	=2985	CALL OVSAP	(ADDRESS)	00	0000+
045F	046B	=2986	JMP CGO	(ADDRESS)	00	0000+
		=2987 ;				
		=2988 ;	COMGOR GO FROM RESET COMMAND			
		=2989 ;	RESET PROCESSOR			
		=2990 ;	RELOAD LOW ORDER PROGRAM BYTES INTO PROGRAM MEMORY			
		=2991 ;				
0461	2302	=2992 COMGOR:	MOV A, #2	(ADDRESS)	00	0000+
0463	3400	=2993	CALL OUTUTL	(ADDRESS)	00	0000+
0465	B910	=2994	ORL P1, #EPRSEI	(ADDRESS)	00	0000+
0467	745A	=2995	CALL OVSAP	(ADDRESS)	00	0000+
0469	99EF	=2996	ANL P1, #(NOT EPRSET)	(ADDRESS)	00	0000+
		=2997 ;				
		=2998 ;				
		=2999 ;	CGO SET UP BREAK LOGIC FOR APPROPRIATE BREAK CONDITIONS,			
		=3000 ;	DEPENDING ON CONTENTS OF 'TYPE'.			
		=3001 ;				
		=3002 CGO:	MMOV A, TYPE	(ADDRESS)	00	0000+
046B	B937	=3011+	MOV R1, #TYPE	(ADDRESS)	00	0000+
046D	F1	=3012+	MOV A, @R1	(ADDRESS)	00	0000+
046E	B371	=3016	ADD A, #LOW GOTEL	(ADDRESS)	00	0000+
0470	B3	=3017	JMPP @A	(ADDRESS)	00	0000+
		=3018 ;				
0471	7C	=3019 GOTEL:	DB LOW(CGONE)	(ADDRESS)	00	0000+

0473 80	=3021	DB	LOW(CG055)				
0474 76	=3022	DB	LOW(CG0PAT)				
0475 80	=3023	DB	LOW(CG0TRA)				
	=3024 ;						
	=3025 CG0PAT:						
0476 99FD	=3026 CG0MB:	ANL	P1, #NOT 00000010B				
0478 8901	=3027	ORL	P1, #00000001B				
047A 8482	=3028	JMP	EPRUN4				
	=3029 ;						
047C 99FC	=3030 CG0MB:	ANL	P1, #NOT 00000011B				
047E 8482	=3031	JMP	EPRUN4				
	=3032 ;						
	=3033 CG0TRA:						
0480 8903	=3034 CG055:	ORL	P1, #00000011B				
	=3035 ;						
	=3036 ; EPRUN4	SET UP	CONTROL LOGIC TO RUN USER'S PROGRAM.				
	=3037 ;		RELEASE PROCESSOR TO RUN.				
	=3038 ;						
0482 8A20	=3039 EPRUN4:	ORL	P2, #00100000B ; DISABLE EP LINK REFERENCES.				
0484 9A1F	=3040	ANL	P2, #NOT 00010000B ; SET ALL REFERENCES TO RAM ARRAY.				
0486 99DF	=3041	ANL	P1, #NOT MODOUT				
0488 F4F4	=3042	CALL	EPREL				
	=3043 ;						
	=3044 ;		WAIT FOR KEYSTROKE INPUT OR HARDWARE BREAK TO OCCUR.				
	=3045 ;						
048A F4AC	=3046 EPRUN1:	CALL	TOFPOL				
048C F4AF	=3047	CALL	KBDPOL				
048E 37	=3048	CPL	A				
048F F295	=3049	JB7	EPRUN3				
0491 8699	=3050	JNI	EPRUN2				
0493 848A	=3051	JMP	EPRUN1				
	=3052 ;						
	=3053 ; EPRUN3	A KEYSTROKE WAS DETECTED WHILE EP WAS RUNNING.					
	=3054 ;		BREAK EXECUTION.				
	=3055 ;		PROCESS KEYSTROKE.				
0495 B400	=3056 EPRUN3:	CALL	STSAVE				
0497 84B3	=3057	JMP	EPRUN5				
	=3058 ;						
	=3059 ; EPRUN2	AN ENABLED BREAK CONDITION OCCURRED.					
	=3060 ;		BREAK EMULATION MODE.				
	=3061 ;		CONTINUE ACCORDING TO GO COMMAND TYPE.				
0499 B400	=3062 EPRUN2:	CALL	STSAVE				
	=3063	MOV	A, TYPE				
049B B937	=3072+	MOV	R1, #TYPE				
049D F1	=3073+	MOV	R1, #R1				
049E 03A1	=3077	ADD	A, #LOW CNTTBL				
04A0 B3	=3078	JMPP	0A				
	=3079 ;						
04A1 A6	=3080 CNTTBL:	DB	LOW(BRKERR)				
04A2 BA	=3081	DB	LOW(EPRUN6)				
04A3 BA	=3082	DB	LOW(EPRUN6)				
04A4 BA	=3083	DB	LOW(CNTTRA)				
04A5 BA	=3084	DB	LOW(CNTTRA)				
	=3085 ;						

LOC	OBJ	LINE	SOURCE STATEMENT	DISASSEMBLY	HEX	DEC
		=3086 ;	BRKERR BREAKPOINT LATCH WAS SET THOUGH BREAKPOINT IS NOT ENABLED.			
		=3087 ;	DISPLAY HARDWARE ERROR MESSAGE.			
04A6	BA0B	=3088	BRKERR: MOV LDATA, #0BH			
04A8	249A	=3089	JMP PERORR			
		=3090 ;				
		=3091	CNTRRA: MMOV A, DSPTIM			
04AA	B928	=3100+	MOV R1, #DSPTIM			
04AC	F1	=3101+	MOV R, @R1			
04AD	94F2	=3105	CALL DELAY			
04AF	F4AF	=3106	CALL KBDPOL			
04B1	F241	=3107	JB7 EPCNT ; B7 SET INDICATES NO KEYSTROKE.			
		=3108 ;				
		=3109 ;	EPRUN5 INPUT(KEY),			
		=3110 ;	IF KEY=END GO TO PARSER,			
		=3111 ;	INPUT KEY,			
		=3112 ;	IF KEY=NEXT GO TO PARSER,			
		=3113 ;	CONTINUE IN SAME MODE.			
		=3114 ;				
04B3	14EC	=3115	EPRUN5: CALL INPKEY			
04B5	FB	=3116	MOV A, KEY			
04B6	D313	=3117	XRL A, #KEYEND			
04B8	96C7	=3118	JNZ EPRET			
04BA	14EC	=3119	EPRUN6: CALL INPKEY			
04BC	FB	=3120	MOV A, KEY			
04BD	D312	=3121	XRL A, #KEYNXT			
04BF	96C7	=3122	JNZ EPRET			
04C1	2302	=3123	MOV A, #2			
04C3	3400	=3124	CALL OUTUTL			
04C5	8441	=3125	JMP EPCNT			
		=3126 ;				
		=3127 ;	EPRET EXECUTION MODE IS TO BE TERMINATED.			
		=3128 ;	JUMP INTO PARSER TO INTERPRET KEY ALREADY DETECTED.			
04C7	0433	=3129	EPRET: JMP MAIN2			
		=3130 ;				
		=3131	SIZECHK			
00C9		=3134+	SIZE SET 201			
		=3135+;				
		=3136+;	*****			
		=3145	\$EJECT			

LOC	OBJ	LINE	SOURCE STATEMENT	DISPATCH	STATUS	TIME	TO
		=3146	CODEBLK 115				
0500		=3171+	ORG 1200				
		=3175 ;	STSAVE EP STATUS SAVE SUBROUTINE.				
		=3176 ;	FORCE CALL TO LOC 014H;				
		=3177 ;	SAVE EP ACC;				
		=3178 ;	SAVE EP TIMER;				
		=3179 ;	SAVE EP PSW;				
		=3180 ;	SAVE EP R0;				
		=3181 ;	SAVE EP TOP-OF-STACK IN EP PC;				
		=3182 ;	RETURN.				
0500	744F	=3183	STSAVE: CALL EPBRK				
0502	2303	=3184	MOV R, #3				
0504	3400	=3185	CALL OUTUTL				
0506	745A	=3186	CALL OVSWAP				
0508	B88F	=3187	MOV R0, #LOW(OV8BAS+OVSIZE)				
050A	746A	=3188	CALL OVLOAD				
050C	8A20	=3189	ORL P2, #00100000B				
050E	2314	=3190	MOV R, #14H				
0510	91	=3191	MOVX @R1, A				
0511	9ADF	=3192	ANL P2, #NOT 00100000B				
0513	8903	=3193	ORL P1, #00000011B				
0515	F4D8	=3194	CALL EPSTEP				
0517	8A20	=3195	ORL P2, #00100000B				
0519	9ADF	=3196	ANL P2, #NOT 00010000B				
051B	8903	=3197	ORL P1, #(ENBRAM OR ENBLNK)				
051D	F4D8	=3198	CALL EPSTEP				
		=3199 ;					
		=3200 ;	EXECUTION PROCESSOR IS NOW AT LOCATION 009H INTERNAL WITH				
		=3201 ;	(RETURN ADDRESS+2) PUSHED ON STACK.				
		=3202 ;					
051F	B895	=3203	MOV R0, #LOW(OV3BAS+OVSIZE)				
0521	746A	=3204	CALL OVLOAD				
0523	F4D0	=3205	CALL EPPASS				
		=3206	MMOV EPACC, A				
0525	B920	=3219+	MOV R1, #EPACC				
0527	A1	=3220+	MOV @R1, A				
0528	F4D0	=3224	CALL EPPASS				
		=3225	MMOV EPTIMR, A				
052A	B922	=3238+	MOV R1, #EPTIMR				
052C	A1	=3239+	MOV @R1, A				
052D	F4D0	=3243	CALL EPPASS				
		=3244	MMOV EPPSW, A				
052F	B921	=3257+	MOV R1, #EPPSW				
0531	A1	=3258+	MOV @R1, A				
0532	F4D0	=3262	CALL EPPASS				
		=3263	MMOV EPR0, A				
0534	B923	=3276+	MOV R1, #EPR0				
0536	A1	=3277+	MOV @R1, A				
0537	B88B	=3281	MOV R0, #LOW(OV1BAS+OVSIZE)				
0539	746A	=3282	CALL OVLOAD				
		=3283	MMOV A, EPPSW				
053B	B921	=3292+	MOV R1, #EPPSW				
053D	F1	=3293+	MOV A, @R1				
053E	07	=3297	DEC A				
053F	5307	=3298	ANL A, #07H				

LOC	OBJ	LINE	SOURCE STATEMENT	THE STATE OF THE	CHG	LOC	OBJ
0541	E7	=3299	RL A				
0542	0308	=3300	ADD A, #08H				
		=3301	MMOV SMALO, A				
0544	B930	=3314+	MOV RL, #SMALO				
0546	A1	=3315+	MOV @RL, A				
0547	F4B7	=3319	CALL EPFET				
0549	03FE	=3320	ADD R, #2				
054B	AA	=3321	MOV LDATA, A				
		=3322	MMOV EPPCLO, A				
054C	B924	=3335+	MOV RL, #EPPCLO				
054E	A1	=3336+	MOV @RL, A				
054F	F4C3	=3340	CALL EPSTOR				
0551	B930	=3341	MOV RL, #SMALO				
0553	11	=3342	INC @R1				
0554	F4B7	=3343	CALL EPFET				
0556	AA	=3344	MOV LDATA, A				
0557	53F0	=3345	ANL A, #11110000B				
0559	2A	=3346	XCH A, LDATA				
055A	13FF	=3347	ADDC A, #-1				
055C	530F	=3348	ANL A, #00001111B				
		=3349	MMOV EPPCHI, A				
055E	B925	=3362+	MOV RL, #EPPCHI				
0560	A1	=3363+	MOV @RL, A				
0561	4A	=3367	ORL A, LDATA				
0562	AA	=3368	MOV LDATA, A				
0563	F4C3	=3369	CALL EPSTOR				
0565	B825	=3370	MOV R0, #EPPCHI				
0567	347C	=3371	CALL UPDAD1				
0569	2340	=3372	MOV A, #01000000B ; "-" FOR DISPLAY				
056B	D4D8	=3373	CALL WDISP				
056D	B820	=3374	MOV R0, #EPACC				
056F	3490	=3375	CALL DSPMID				
0571	03	=3376	RET				
		=3377	SIZECHK				
0072		=3380+ SIZE SET 114					
		=3381+;					
		=3382+; *****					
		=3391 #EJECT					

	3392 +	INCLUDE: PG. FILE: RW7					
0000	=3393	CHARCR EQU 0DH ;CCR					
000A	=3394	CHARLF EQU 0AH ;CLF					
001A	=3395	CNTRLZ EQU 1AH ;CONTROL-Z					
	=3396 ;						
	=3397	CODEBLK 80					
0297	=3412+	ORG 663					
	=3416	;HRECIN HEXFILE RECORD INPUT ROUTINE					
0297 34CD	=3417	HRECIN: CALL CHARIN					
0299 D31A	=3418	XRL A, #CNTRLZ					
029B C6E0	=3419	JZ DONE					
029D D31A	=3420	XRL A, #CNTRLZ					
029F D33A	=3421	XRL A, #'(')					
02A1 9697	=3422	JNZ HRECIN					
	=3423	MMOV CHKSUM, ZERO					
02A3 BD00	=3424+	MOV CHKSUM, #ZERO					
02A5 14F0	=3432	CALL BYTEIN					
	=3433	MMOV BUFcnt, A					
02A7 B941	=3446+	MOV R1, #BUFCNT					
02A9 A1	=3447+	MOV @R1, A					
02AA 14F0	=3451	CALL BYTEIN					
	=3452	MMOV SMAHI, A					
02AC B931	=3465+	MOV R1, #SMAHI					
02AE A1	=3466+	MOV @R1, A					
02AF 14F0	=3470	CALL BYTEIN					
	=3471	MMOV SMALO, A					
02B1 B930	=3484+	MOV R1, #SMALO					
02B3 A1	=3485+	MOV @R1, A					
02B4 14F0	=3489	CALL BYTEIN					
	=3490	MMOV RECTYP, A					
02B6 B942	=3503+	MOV R1, #RECTYP					
02B8 A1	=3504+	MOV @R1, A					
	=3508 ;						
	=3509	;HDATIN HEX DATA BYTE IN					
	=3510	HDATIN: MMOV A, BUFcnt					
02B9 B941	=3519+	MOV R1, #BUFCNT					
02BB F1	=3520+	MOV A, @R1					
02BC C6CC	=3524	JZ RECDON					
02BE 14F0	=3525	CALL BYTEIN					
02C0 AA	=3526	MOV LDATA, A					
02C1 F400	=3527	CALL LSTORE					
02C3 34F2	=3528	CALL INCSMA					
	=3529	MDEC BUFcnt					
02C5 B941	=3534+	MOV R1, #BUFCNT					
02C7 F1	=3535+	MOV A, @R1					
02C8 07	=3539+	DEC A					
02C9 A1	=3544+	MOV @R1, A					
02CA 44B9	=3547	JMP HDATIN					
	=3548 ;						
02CC 34CD	=3549	RECDON: CALL CHARIN					
02CE D33F	=3550	XRL A, #'(')					
02D0 C6DB	=3551	JZ CKSMOK					
02D2 D33F	=3552	XRL A, #'(')					
02D4 34BA	=3553	CALL NIBIN2					
02D6 14F2	=3554	CALL BYTE11					

; SWITCH BACK TO DATA CHARACTER
; JOIN SUBROUTINE ALREADY IN PROGRESS
; DITTO

LOC	OBJ	LINE	SOURCE STATEMENT	SYMBOL TABLE	DISPATCH	LOC	OBJ
		=3555					
		=3556					
		=3557	MMOV A, CHKSUM				
02D8	FD	=3573+	MOV A, CHKSUM				
02D9	96E1	=3577	JNZ CHKERR				
		=3578	CKSMOK: MMOV A, RECTYP				
02DB	B942	=3587+	MOV R1, #RECTYP				
02DD	F1	=3588+	MOV A, @R1				
02DE	C697	=3592	JZ HRECIN				
		=3593 ;					
		=3594 ;	DONE HEX FILE CORRECTLY RECEIVED				
02E0	83	=3595	DONE: RET				
		=3596 ;					
		=3597 ;	CHKERR CHECKSUM ERROR IN INPUT RECORD DETECTED				
02E1	BA0C	=3598	CHKERR: MOV LDATA, #0CH				
02E3	249A	=3599	JMP PLERROR				
		=3600	SIZECHK				
004E		=3603+	SIZE SET 78				
		=3604+;					
		=3605+;	*****				
		=3614 ;					
		=3615	CODEBLK 12				
00F0		=3620+	ORG 240				
		=3624 ;	BYTEIN BYTE INPUT SUBROUTINE.				
		=3625 ;	RECEIVES TWO HEXIDECIMAL CHARACTERs FROM THE TAPE INPUT DEVICE				
		=3626 ;	AND ASSEMBLES THEM INTO A SINGLE BYTE OF DATA.				
00F0	34B8	=3627	BYTEIN: CALL NIBIN				
00F2	47	=3628	BYTE1: SWAP A				
00F3	AA	=3629	MOV LDATA, A				
00F4	34B8	=3630	CALL NIBIN				
		=3631	MORL LDATA, A				
00F6	4A	=3640+	ORL A, LDATA				
00F7	AA	=3660+	MOV LDATA, A				
00F8	6D	=3664	ADD A, CHKSUM				
00F9	AD	=3665	MOV CHKSUM, A				
00FA	FA	=3666	MOV A, LDATA				
00FB	83	=3667	RET				
		=3668	SIZECHK				
00FC		=3671+	SIZE SET 12				
		=3672+;					
		=3673+;	*****				
		=3682 ;					
		=3683	CODEBLK 25				
01B8		=3693+	ORG 440				
		=3697 ;	NIBIN RECEIVES A HEXIDECIMAL CHARACTER AND PRODUCES A MASKED FOUR BIT VALUE.				
		=3698 ;	NOTE- ERROR CHECKING DONE TO VERIFY HEXIDECIMAL VALIDITY				
01B8	34CD	=3699	NIBIN: CALL CHARIN				
01BA	03C6	=3700	NIBIN2: ADD A, #-3AH ; ACC=0F6-0FF FOR CHARACTERs '0'-'9'				
		=3701	; CHARACTERs > '9' PRODUCE OVERFLOW				
01BC	E6C2	=3702	JNC NIBI3				
01BE	03F9	=3703	ADD A, #-7 ; ACC=0-5 FOR CHARACTERs 'A'-'F'				
01C0	E6C9	=3704	JNC ASCERR ; ERROR IF CHARACTER BETWEEN '9' AND 'A'				
		=3705 ;					
		=3706 ;	ACC=0F61-05H FOR CHARACTERs '0'-'F'				
		=3707 ;					

LOC	OBJ	LINE	SOURCE STATEMENT	THIRDATE 20002	001	000
01C2	03FA	=3708	NIB13: ADD A, #6 ; ACC=0F0H-0FFH FOR CHARACTERS '0'-'F'			
01C4	0310	=3709	ADD A, #10H ; ACC=00H-0FH FOR CHARACTERS '0'-'F';			
		=3710	; OVERFLOW IF ABOVE IS TRUE.			
01C6	E6C9	=3711	JNC ASCERR			
01C8	83	=3712	RET			
		=3713 ;				
		=3714 ;	ASCERR ILLEGAL HEXIDECIMAL CHARACTER RECEIVED			
01C9	8A0A	=3715	ASCERR: MOV LDATA, #0AH			
01CB	249A	=3716	JMP PERROR			
		=3717	SIZECHK			
0015		=3720+	SIZE SET 21			
		=3721+;				
		=3722+;	*****			
		=3731 ;				
		=3732 ;				
		=3733	CODEBLK 5			
01CD		=3743+	ORG 461			
		=3747 ;	CHARIN CHARACTER INPUT ROUTINE.			
		=3748 ;	RECEIVES ONE ASCII CHARACTER FROM THE LOGICAL READER DEVICE.			
01CD	D449	=3749	CHARIN: CALL CIN			
01CF	537F	=3750	ANL A, #7FH			
01D1	83	=3751	RET			
		=3752	SIZECHK			
0005		=3755+	SIZE SET 5			
		=3756+;				
		=3757+;	*****			
		=3766 ;				
		=3767 ;				
		=3768	#EJECT			

LOC	OBJ	LINE	SOURCE STATEMENT	SYMBOL TABLE	ADDRESS	VALUE	HEX	DEC
		=3769	CODEBLK 100					
0572		=3794+	ORG 1394					
		=3798	;HFILED HEX FILE OUTPUT SUBROUTINE					
		=3799	; WHEN CALLED WITH F0=0 OUTPUT IS STANDARD HEX FILE FORMAT.					
		=3800	; WHEN CALLED WITH F0=1 OUTPUT IS FORMATTED DATA DUMP TO CRT.					
		=3801	HFILED: MMOV MEMHI, SMAHI					
0572 B931		=3817+	MOV R1, #SMAHI					
0574 F1		=3818+	MOV R1, @R1					
0575 B935		=3824+	MOV R1, #MEMHI					
0577 A1		=3825+	MOV @R1, A					
		=3826	MMOV MEMLO, SMALO					
0578 B938		=3844+	MOV R1, #SMALO					
057A F1		=3845+	MOV R1, @R1					
057B B934		=3851+	MOV R1, #MEMLO					
057D A1		=3852+	MOV @R1, A					
		=3855	MMOV CHKSUM, ZERO					
057E B000		=3860+	MOV CHKSUM, #ZERO					
0580 B865		=3864	MOV R0, #HEXBUFF					
		=3865	;					
		=3866	;LDBYTE LOAD NEXT BYTE FROM MEMORY INTO HEX BUFFER					
0582 14FC		=3867	LDBYTE: CALL LFETCH					
0584 FA		=3868	MOV A, LDATA					
0585 A0		=3869	MOV @R0, A					
0586 18		=3870	INC R0					
0587 B4E2		=3871	CALL CMPHAS					
0589 E696		=3872	JNC ENDFIL					
058B 34F2		=3873	CALL INCSMA					
058D F8		=3874	MOV A, R0					
058E 0388		=3875	ADD A, #- (BUFLN+HEXBUFF)					
0590 E682		=3876	JNC LDBYTE					
0592 D400		=3877	CALL HREC0					
0594 A472		=3878	JMP HFILED					
		=3879	;					
		=3880	;ENDFIL END HEX FILE TRANSMISSION:					
		=3881	; PRINT OUT BUFFER FOR LAST DATA RECORD					
		=3882	; PRINT OUT CANNED 'END-OF-FILE' RECORD					
		=3883	; RETURN.					
0596 D400		=3884	ENDFIL: CALL HREC0					
0598 B6A7		=3885	JF0 HFDONE					
059A 34D2		=3886	CALL TCRLFO					
059C B8AE		=3887	MOV R0, # (LOW EOFREC)					
059E F8		=3888	ENDF1: MOV A, R0					
059F A3		=3889	MOV @R0, A					
05A0 C6A7		=3890	JZ HFDONE					
05A2 B4B0		=3891	CALL CHAR0					
05A4 18		=3892	INC R0					
05A5 A49E		=3893	JMP ENDF1					
05A7 34D2		=3894	HFDONE: CALL TCRLFO					
05A9 231A		=3895	MOV A, #CNTRLZ					
05AB B4B0		=3896	CALL CHAR0					
05AD 83		=3897	RET					
		=3898	;					
		=3899	;EOFREC CHARACTER STRING FOR CANNED END-OF-FILE RECORD FOR					
		=3900	; INTEL HEX FILE FORMAT STANDARD.					
05AE 203A3830		=3901	EOFREC: DB '0000001FF'					

0000 00	=3902	DB	0	; END OF STRING CODE BYTE
	=3903	SIZECHK		
0049	=3904	SIZE SET	73	
	=3907+			
	=3908+;	*****		
	=3917 ;			
	=3918 ;			
	=3919	CODEBLK	90	
0600	=3949+	ORG	1536	
	=3953 ;	HRECO		HEXIDEcimal RECORD OUTPUT SEQUENCE.
	=3954 ;			HEX BUFFER ALREADY LOADED.
0600 F8	=3955 HRECO:	MOV	A, R0	
0601 039B	=3956	ADD	A, #-HEXBUF	
	=3957	MMOV	BUFCNT, A	
0603 B941	=3970+	MOV	R1, #BUFCNT	
0605 A1	=3971+	MOV	0R1, A	
0606 34D2	=3975	CALL	TCRLF0	
0608 2320	=3976	MOV	A, #' '	
060A B4BD	=3977	CALL	CHAR0	
060C D617	=3978	JF0	FDUMP1	
060E 233A	=3979	MOV	A, #' '	
0610 B4BD	=3980	CALL	CHAR0	
	=3981	MMOV	A, BUFCNT	
0612 B941	=3990+	MOV	R1, #BUFCNT	
0614 F1	=3991+	MOV	A, 0R1	
0615 34DB	=3995	CALL	BYTE0	
	=3996 FDUMP1:	MMOV	A, MEMHI	
0617 B935	=4005+	MOV	R1, #MEMHI	
0619 F1	=4006+	MOV	A, 0R1	
061A 34DB	=4010	CALL	BYTE0	
	=4011	MMOV	A, MEMLO	
061C B934	=4020+	MOV	R1, #MEMLO	
061E F1	=4021+	MOV	A, 0R1	
061F 34DB	=4025	CALL	BYTE0	
0621 B628	=4026	JF0	FDUMP2	
0623 27	=4027	CLR	A	
0624 34DB	=4028	CALL	BYTE0	
0626 C42C	=4029	JMP	DAT0	
0628 233D	=4030 FDUMP2:	MOV	A, #'='	
062A B4BD	=4031	CALL	CHAR0	
	=4032 ;	DAT0		DATA OUTPUT
062C B865	=4033 DAT0:	MOV	R0, #HEXBUF	
062E B632	=4034 DAT01:	JF0	FDUMP5	
0630 C436	=4035	JMP	FDUMP3	
0632 2320	=4036 FDUMP5:	MOV	A, #' '	
0634 B4BD	=4037	CALL	CHAR0	
0636 F0	=4038 FDUMP3:	MOV	A, 0R0	
0637 34DB	=4039	CALL	BYTE0	
0639 18	=4040	INC	R0	
	=4041	MDJNZ	BUFCNT, DAT01	
063A B941	=4046+	MOV	R1, #BUFCNT	
063C F1	=4047+	MOV	A, 0R1	
063D 07	=4051+	DEC	A	

LOC	OBJ	LINE	SOURCE STATEMENT	TRANSLATED STATEMENT	LOC	OBJ
063E	A1	=4056+	MOV @R1, A	MOV R1, A	063E	A1
063F	962E	=4060+	JNZ DAT01	JNZ 0000	063F	962E
		=4062 ;				
		=4063 ;	ENDREC: END RECORD BEING TRANSMITTED			
0641	B648	=4064	ENDREC: JF0 FDUMP4	JF0 0000	0641	B648
		=4065	MNOV A, CHKSUM	MNOV A, 0000		
0643	FD	=4081+	MOV A, CHKSUM	MOV A, 0000	0643	FD
0644	37	=4085	CPL A	CPL A	0644	37
0645	17	=4086	INC A	INC A	0645	17
0646	34DB	=4087	CALL BYTE0	CALL 0000	0646	34DB
0648	83	=4088	FDUMP4: RET	RET	0648	83
		=4089	SIZECHK			
0049		=4092+	SIZE SET 73	SIZE 73	0049	
		=4093+;				
		=4094+;	*****			
		=4103 ;				
		=4104	CODEBLK 9			
01D2		=4114+	ORG 466	ORG 466	01D2	
		=4118 ;	TCRLF0 TAPE <CR><LF> OUTPUT			
01D2	230D	=4119	TCRLF0: MOV A, #CHARCR	MOV A, 0000	01D2	230D
01D4	B4BD	=4120	CALL CHAR0	CALL 0000	01D4	B4BD
01D6	230A	=4121	MOV A, #CHARLF	MOV A, 0000	01D6	230A
01D8	B4BD	=4122	CALL CHAR0	CALL 0000	01D8	B4BD
01DA	83	=4123	RET	RET	01DA	83
		=4124	SIZECHK			
0009		=4127+	SIZE SET 9	SIZE 9	0009	
		=4128+;				
		=4129+;	*****			
		=4138 ;				
		=4139	CODEBLK 11			
01DB		=4149+	ORG 475	ORG 475	01DB	
		=4153 ;	BYTE0 BYTE OUTPUT			
01DB	AA	=4154	BYTE0: MOV LDATA, A	MOV LDATA, A	01DB	AA
01DC	6D	=4155	ADD A, CHKSUM	ADD A, 0000	01DC	6D
01DD	AD	=4156	MOV CHKSUM, A	MOV 0000, A	01DD	AD
01DE	FA	=4157	MOV A, LDATA	MOV A, 0000	01DE	FA
01DF	47	=4158	SWAP A	SWAP A	01DF	47
01E0	B4BB	=4159	CALL NIB0	CALL 0000	01E0	B4BB
01E2	FA	=4160	MOV A, LDATA	MOV A, 0000	01E2	FA
01E3	B4BB	=4161	CALL NIB0	CALL 0000	01E3	B4BB
01E5	83	=4162	RET	RET	01E5	83
		=4163	SIZECHK			
000B		=4166+	SIZE SET 11	SIZE 11	000B	
		=4167+;				
		=4168+;	*****			
		=4177 ;				
		=4178	CODEBLK 12			
01E6		=4188+	ORG 486	ORG 486	01E6	
		=4192 ;	HEXRASC: HEXIDECIMAL NIBBLE TO ASCII CHARACTER CONVERSION.			
01E6	530F	=4193	HEXRASC: ANL A, #0FH	ANL A, 000F	01E6	530F
01E8	03F6	=4194	ADD A, #(-10)	ADD A, -10	01E8	03F6
01EA	F6EF	=4195	JC HEXNIB	JC 0000	01EA	F6EF
01EC	033A	=4196	ADD A, #(10+'0')	ADD A, 30	01EC	033A
01EE	83	=4197	RET	RET	01EE	83
01EF	0341	=4198	HEXNIB: ADD A, #'A'	ADD A, 41	01EF	0341

LOC	OBJ	LINE	SOURCE STATEMENT	DISPATCH	TIME	WQ	DO
01F1	83	=4199	RET				
		=4200	SIZECHK				
000C		=4203+	SIZE SET 12				
		=4204+;					
		=4205+; *****					
		=4214 ;					
		=4215 ;					
		=4216	DECLARE BITSO,CONST				
000B		=4230	BITSO EQU 11 ;DATA BITS PUT OUT (INCLUDING TWO STOP BITS)				
		=4231 ;					
		=4232	CODEBLK 30				
04C9		=4252+	ORG 1225				
		=4256	;HBDLAY HALF-BIT TIME DELAY				
		=4257	HBDLAY: MMOV H,HBITHI				
04C9 B927		=4273+	MOV R1,HBITHI				
04CB F1		=4274+	MOV A,0R1				
04CC B945		=4280+	MOV R1,#H				
04CE A1		=4281+	MOV 0R1,A				
		=4284	MMOV R1,HBITLO				
04CF B926		=4300+	MOV R1,HBITLO				
04D1 F1		=4301+	MOV A,0R1				
04D2 A9		=4314+	MOV R1,A				
04D3 84D7		=4317	JMP HBD1				
04D5 B900		=4318	HBD2: MOV R1,#0				
04D7 E9D7		=4319	HBD1: DJNZ R1,HBD1				
		=4320	MDJNZ H,HBD2				
04D9 B945		=4325+	MOV R1,#H				
04DB F1		=4326+	MOV A,0R1				
04DC 07		=4330+	DEC A				
04DD A1		=4335+	MOV 0R1,A				
04DE 96D5		=4339+	JNZ HBD2				
04E0 83		=4341	RET				
		=4342	SIZECHK				
0018		=4345+	SIZE SET 24				
		=4346+;					
		=4347+; *****					
		=4356 ;					
		=4357	;\$EJECT				

LOC	OBJ	LINE	SOURCE STATEMENT	DISPATCH	300000	300000	300000
		=4358	CODEBLK 40				
058B		=4383+	ORG 1467				
		=4387 ;NIBO	MASK ACC TO MAKE HEX NIBBLE, TRANSLATE TO ASCII AND OUTPUT				
058B 34E6		=4388 NIBO:	CALL HEXASC				
		=4389 ;					
		=4390 ;CHARO	CONSOLE OUTPUT SUBROUTINE				
		=4391 ;	WRITES THE CONTENTS OF THE ACC TO THE CRT DISPLAY SCREEN				
		=4392 CHARO:	MNOV REGC, A				
058D B944		=4405+	MOV R1, #REGC				
05BF A1		=4406+	MOV @R1, A				
		=4410	MNOV B, BIT50 ; SET NUMBER OF BITS TO BE TRANSMITTED				
05C0 B943		=4421+	MOV R1, #B				
05C2 B10B		=4422+	MOV @R1, #BIT50				
05C4 97		=4426	CLR C ; CLEAR CARRY				
05C5 F6CB		=4427 C01:	JC C02				
05C7 99BF		=4428	RNLT P1, #NOT TTYOUT				
05C9 A4CF		=4429	JMP C03				
05CB 8940		=4430 C02:	ORL P1, #TTYOUT				
05CD 00		=4431	NOP ; EVEN OUT TWO BRANCH EXECUTION TIMES				
05CE 00		=4432	NOP				
05CF 94C9		=4433 C03:	CALL HBDLAY				
05D1 94C9		=4434	CALL HBDLAY				
05D3 97		=4435	CLR C ; SET WHAT WILL EVENTUALLY BECOME A STOP BIT				
05D4 A7		=4436	CPL C				
		=4437	MRRC REGC ; ROTATE CHARACTER RIGHT ONE BIT,				
05D5 B944		=4442+	MOV R1, #REGC				
05D7 F1		=4443+	MOV A, @R1				
05D8 67		=4447+	RRC A				
05D9 A1		=4452+	MOV @R1, A				
		=4455	; \ MOVING NEXT DATA BIT INTO CARRY				
		=4456	MDJNZ B, C01 ; CHECK IF CHARACTER (AND STOP BIT(S)) DONE				
05DA B943		=4461+	MOV R1, #B				
05DC F1		=4462+	MOV A, @R1				
05DD 07		=4466+	DEC A				
05DE A1		=4471+	MOV @R1, A				
05DF 96C5		=4475+	JNZ C01				
05E1 83		=4477	RET				
		=4478	SIZECHK				
0027		=4481+ SIZE	SET 39				
		=4482+;					
		=4483+;*****					
		=4492 ;					
		=4493	CODEBLK 47				
0649		=4523+	ORG 1609				
		=4527 ;CIN	CONSOLE INPUT SUBROUTINE WAITS FOR A KEYSTROKE AND				
		=4528 ;	RETURNS WITH 8 BITS IN REG ACC.				
0649 B943		=4529 CIN:	MOV R1, #B				
064B B10B		=4530	MOV @R1, #B ; DATA BITS TO BE READ				
064D 464D		=4531 C10:	JNT1 C10				
064F 464D		=4532	JNT1 C10				
0651 5651		=4533 C11:	JT1 C11				
0653 5651		=4534	JT1 C11				
0655 94C9		=4535	CALL HBDLAY				
0657 5651		=4536	JT1 C11				
0659 94C9		=4537 C12:	CALL HBDLAY				

065F 97	=4540	CLR	C	DATA BIT IN CY	0000	0000	0000	0000
0660 C465	=4541	JMP	C14		0000	0000	0000	0000
0662 97	=4542 C13:	CLR	C		0000	0000	0000	0000
0663 A7	=4543	CPL	C		0000	0000	0000	0000
0664 00	=4544	NOP		EVEN OUT BRANCH EXECUTION TIMES	0000	0000	0000	0000
0665 00	=4545 C14:	NOP			0000	0000	0000	0000
0666 00	=4546	NOP			0000	0000	0000	0000
0667 00	=4547	NOP			0000	0000	0000	0000
	=4548	MRRC	REGC		0000	0000	0000	0000
0668 B944	=4553+	MOV	RL, #REGC		0000	0000	0000	0000
066A F1	=4554+	MOV	A, @R1		0000	0000	0000	0000
066B 67	=4558+	RRC	A		0000	0000	0000	0000
066C A1	=4563+	MOV	@R1, A		0000	0000	0000	0000
	=4566	MDJNZ	B, C12		0000	0000	0000	0000
066D B943	=4571+	MOV	RL, #B		0000	0000	0000	0000
066F F1	=4572+	MOV	A, @R1		0000	0000	0000	0000
0670 07	=4576+	DEC	A		0000	0000	0000	0000
0671 A1	=4581+	MOV	@R1, A		0000	0000	0000	0000
0672 9659	=4585+	JNZ	C12		0000	0000	0000	0000
	=4587	MNOV	A, REGC		0000	0000	0000	0000
0674 B944	=4596+	MOV	RL, #REGC		0000	0000	0000	0000
0676 F1	=4597+	MOV	A, @R1		0000	0000	0000	0000
0677 83	=4601	RET		CHARACTER COMPLETE	0000	0000	0000	0000
	=4602	SIZECHK			0000	0000	0000	0000
067F	=4605+ SIZE	SET	47		0000	0000	0000	0000
	=4606+				0000	0000	0000	0000
	=4607+;				0000	0000	0000	0000
	=4616 \$EJECT				0000	0000	0000	0000

LOC	OBJ	LINE	SOURCE STATEMENT	SYMBOLS	VALUES	LOC	OBJ
		4617 \$	INCLUDE(:F0:MEMREF.MOD)				
		=4618	CODEBLK 15				
02E5		=4633+	ORG 741				
		=4637 ;	CONFIL COMMAND TO FILL ADDRESS SPACE BETWEEN SMA AND EMA WITH DATA				
		=4638 ;	IN LOW BYTE OF MEM.				
		=4639 CONFIL:	MMOV LDATA, MEMLO				
02E5 B934		=4655+	MOV R1, #MEMLO				
02E7 F1		=4656+	MOV A, @R1				
02E8 AA		=4669+	MOV LDATA, A				
02E9 F400		=4672 LFILL:	CALL LSTORE				
02EB B4E2		=4673	CALL CMPHAS				
02ED E6F3		=4674	JNC LFILL1				
02EF 34F2		=4675	CALL INC SMA				
02F1 44E9		=4676	JMP LFILL				
02F3 83		=4677 LFILL1:	RET				
		=4678	SIZECHK				
000F		=4681+ SIZE	SET 15				
		=4682+;					
		=4683+;	*****				
		=4692 ;					
		=4693	CODEBLK 4				
00FC		=4698+	ORG 252				
		=4702 ;	LFETCH FETCHES CONTENTS OF LOGICAL MEMORY ADDRESS DETERMINED BY				
		=4703 ;	<TYPE>, <SMAHI>, & <SMALO> INTO <LDATA>.				
00FC D478		=4704 LFETCH:	CALL AFETCH				
00FE AA		=4705	MOV LDATA, A				
00FF 83		=4706	RET				
		=4707	SIZECHK				
0004		=4710+ SIZE	SET 4				
		=4711+;					
		=4712+;	*****				
		=4721 ;					
		=4722	CODEBLK 75				
0678		=4752+	ORG 1656				
		=4756 ;					
		=4757 ;	AFETCH LOGICAL FETCH SUBROUTINE				
		=4758 ;	FETCHS CONTENTS OF VARIOUS MEMORY SPACES TO ACC.				
		=4759 AFETCH:	MMOV A, TYPE				
0678 B937		=4768+	MOV R1, #TYPE				
067A F1		=4769+	MOV A, @R1				
067B 037E		=4773	ADD A, #LOW LFETBL				
067D B3		=4774	JMP @A				
		=4775 ;					
067E 84		=4776 LFETBL:	DB LOW LFEPM				
067F 98		=4777	DB LOW LFEDM				
0680 9C		=4778	DB LOW LFEREG				
0681 A9		=4779	DB LOW LFEINT				
0682 B1		=4780	DB LOW LFEBRK				
0683 B1		=4781	DB LOW LFEBRK				
		=4782 ;					
		=4783 LFEPM:	MMOV A, SMAHI				
0684 B931		=4792+	MOV R1, #SMAHI				
0686 F1		=4793+	MOV A, @R1				
0687 9698		=4797	JNZ LFEDM				
		=4798	MMOV A, SMALO				

LOC	OBJ	LINE	SOURCE STATEMENT	SYMBOL TABLE	DISASSEMBLY	LOC	OBJ
0689	B930	=4887+	MOV R1, #SMALO				
068B	F1	=4888+	MOV A, @R1				
068C	03E9	=4812	ADD A, #-OVSZ				
068E	F698	=4813	JC LFEDM				
		=4814	MMOV A, SMALO				
0690	B930	=4823+	MOV R1, #SMALO				
0692	F1	=4824+	MOV A, @R1				
0693	034E	=4826	ADD A, #OVBUF				
0695	A9	=4829	MOV R1, A				
0696	F1	=4830	MOV A, @R1				
0697	83	=4831	RET				
0698	94E1	=4832 LFEDM:	CALL LPGSEL				
069A	81	=4833	MOVX A, @R1				
069B	83	=4834	RET				
		=4835 ;					
		=4836 LFEREC:	MMOV A, SMALO				
069C	B930	=4845+	MOV R1, #SMALO				
069E	F1	=4846+	MOV A, @R1				
069F	537F	=4850	ANL A, #01111111B ; CHECK IF LOW 7 BITS =0				
06A1	C6A5	=4851	JZ LFER0				
06A3	E4B7	=4852	JMP EPFET				
		=4853 ;					
		=4854 LFER0:	MMOV A, EP0				
06A5	B923	=4863+	MOV R1, #EP0				
06A7	F1	=4864+	MOV A, @R1				
06A8	83	=4868	RET				
		=4869 ;					
		=4870 LFEINT:	MMOV A, SMALO				
06A9	B930	=4879+	MOV R1, #SMALO				
06AB	F1	=4880+	MOV A, @R1				
06AC	0320	=4884	ADD A, #EPACC				
06AE	A9	=4885	MOV R1, A				
06AF	F1	=4886	MOV A, @R1				
06B0	83	=4887	RET				
		=4888 ;					
		=4889 ; LFEBRK LOGICAL FETCH OF BREAK-POINT DATA					
06B1	94E1	=4890 LFEBRK:	CALL LPGSEL				
06B3	99F7	=4891	ANL P1, #NOT 00001000B				
06B5	8908	=4892	ORL P1, #00001000B				
06B7	99FD	=4893	ANL P1, #NOT 00000010B				
06B9	8901	=4894	ORL P1, #00000001B				
06BB	81	=4895	MOVX A, @R1				
06BC	2301	=4896	MOV A, #01H				
06BE	86C1	=4897	JNI LFEBR1				
06C0	27	=4898	CLR A				
06C1	83	=4899 LFEBR1:	RET				
		=4900	SIZECHK				
004A		=4903+ SIZE SET 74					
		=4904+;					
		=4905+; *****					
		=4914 \$EJECT					

LOC	OBJ	LINE	SOURCE STATEMENT	INSTR	OP	PC	PC+1
		=4915	CODEBLK 85				
0700		=4950+	ORG 1792				
		=4954 ;					
		=4955 ;	LSTORE LOGICAL STORE SUBROUTINE				
		=4956 ;	STORES CONTENTS OF LDATA INTO VARIOUS MEMORY SPACES.				
		=4957 LSTORE:	MMOV A, TYPE				
0700 B937		=4966+	MOV R1, #TYPE				
0702 F1		=4967+	MOV A, @R1				
0703 0306		=4971	ADD A, #LOW LSTBL				
0705 B3		=4972	JMPP @A				
		=4973 ;					
0706 0C		=4974 LSTBL:	DB LOW LSTPM				
0707 21		=4975	DB LOW LSTDM				
0708 26		=4976	DB LOW LSTREG				
0709 34		=4977	DB LOW LSTINT				
070A 3D		=4978	DB LOW LSTBRK				
070B 3D		=4979	DB LOW LSTBRK				
		=4980 ;					
		=4981 LSTPM:	MMOV A, SMAHI				
070C B931		=4990+	MOV R1, #SMAHI				
070E F1		=4991+	MOV A, @R1				
070F 9621		=4995	JNZ LSTDM				
		=4996	MMOV A, SMALO				
0711 B930		=5005+	MOV R1, #SMALO				
0713 F1		=5006+	MOV A, @R1				
0714 03E9		=5010	ADD A, #OVSZ				
0716 F621		=5011	JC LSTDM				
		=5012	MMOV A, SMALO				
0718 B930		=5021+	MOV R1, #SMALO				
071A F1		=5022+	MOV A, @R1				
071B 034E		=5026	ADD A, #OVBUF				
071D A9		=5027	MOV R1, A				
071E FA		=5028	MOV A, LDATA				
071F A1		=5029	MOV @R1, A				
0720 83		=5030	RET				
		=5031 ;					
0721 94E1		=5032 LSTDM:	CALL LPGSEL				
0723 FA		=5033	MOV A, LDATA				
0724 91		=5034	MOVX @R1, A				
0725 83		=5035	RET				
		=5036 ;					
		=5037 LSTREG:	MMOV A, SMALO				
0726 B930		=5046+	MOV R1, #SMALO				
0728 F1		=5047+	MOV A, @R1				
0729 537F		=5051	ANL A, #01111111B ; CHECK IF LOW ORDER BITS = 0				
072B C62F		=5052	JZ LSTR0				
072D E4C3		=5053	JMP EPSTOR				
		=5054 ;					
		=5055 LSTR0:	MMOV EPR0, LDATA				
072F FA		=5078+	MOV A, LDATA				
0730 B923		=5084+	MOV R1, #EPR0				
0732 A1		=5085+	MOV @R1, A				
0733 83		=5088	RET				
		=5089 ;					
		=5090 LSTINT:	MMOV A, SMALO				

0730 F1	=5100+	MOV	A, @R1				
0737 0320	=5104	ADD	A, #EPACC				
0739 A9	=5105	MOV	R1, A				
073A FA	=5106	MOV	A, LDATA				
073B A1	=5107	MOV	@R1, A				
073C 83	=5108	RET					
	=5109 ;						
	=5110 ; LSTBRK LOGICAL STORE OF BREAK-POINT DATA						
073D 94E1	=5111 LSTBRK:	CALL	LPGSEL				
073F FA	=5112	MOV	A, LDATA				
0740 1246	=5113	JB0	LSTBR1				
0742 8901	=5114	ORL	P1, #00000001B				
0744 E448	=5115	JMP	LSTBR2				
0746 99FE	=5116 LSTBR1:	ANL	P1, #NOT 00000001B				
0748 99F7	=5117 LSTBR2:	ANL	P1, #NOT 00001000B				
074A 81	=5118	MOVX	A, @R1				
074B 8908	=5119	ORL	P1, #00001000B				
074D 83	=5120	RET					
	=5121	SIZECHK					
004E	=5124+ SIZE SET 78						
	=5125+;						
	=5126+; *****						
	=5135 ;						
	=5136	CODEBLK 17					
04E1	=5156+ ORG 1249						
	=5160 ; LPGSEL LOGICAL PAGE SELECT.						
	=5161 ; SETS UP PORT 2 TO ADDRESS APPROPRIATE BYTE OF RAM BLOCK.						
	=5162 LPGSEL:	MMOV	A, TYPE				
04E1 B937	=5171+	MOV	R1, #TYPE				
04E3 F1	=5172+	MOV	A, @R1				
04E4 5301	=5176	ANL	A, #00000001B				
04E6 47	=5177	SWAP	A				
	=5178	MORL	A, SMAHI				
04E7 B931	=5104+	MOV	R1, #SMAHI				
04E9 41	=5185+	ORL	A, @R1				
04EA 4340	=5189	ORL	A, #01000000B				
04EC 3A	=5190	OUTL	P2, A				
	=5191	MMOV	A, SMALO				
04ED B930	=5200+	MOV	R1, #SMALO				
04EF F1	=5201+	MOV	A, @R1				
04F0 A9	=5205	MOV	R1, A				
04F1 83	=5206	RET					
	=5207	SIZECHK					
0011	=5210+ SIZE SET 17						
	=5211+;						
	=5212+; *****						
	=5221 ;						
	=5222 \$EJECT						

LOC	OBJ	LINE	SOURCE STATEMENT	HEX STATE	ADDR	DISP	LEN	VAL
		=5223	CODEBLK 11					
01F2		=5233+	ORG 498					
		=5237	INCSMA INCREMENT STARTING MEMORY ADDRESS WORD.					
01F2 B930		=5238	INCSMA: MOV R1, #SMALO					
01F4 11		=5239	INCH: INC @R1					
01F5 F1		=5240	MOV A, @R1					
01F6 96FC		=5241	JNZ INCH1					
01F8 19		=5242	INC R1					
01F9 F1		=5243	MOV A, @R1					
01FA 17		=5244	INC A					
01FB 31		=5245	XCHD A, @R1					
01FC 83		=5246	INCH1: RET					
		=5247	SIZECHK					
000B		=5250+	SIZE SET 11					
		=5251+;						
		=5252+;	*****					
		=5261 ;						
		=5262	CODEBLK 12					
02F4		=5277+	ORG 756					
		=5281	DECSMA DECREMENT SMA WORD.					
02F4 B930		=5282	DECSMA: MOV R1, #SMALO					
02F6 F1		=5283	MOV A, @R1					
02F7 07		=5284	DEC A					
02F8 21		=5285	XCH A, @R1					
02F9 96FF		=5286	JNZ DECSM1					
02FB 19		=5287	INC R1					
02FC F1		=5288	MOV A, @R1					
02FD 07		=5289	DEC A					
02FE 31		=5290	XCHD A, @R1					
02FF 83		=5291	DECSM1: RET					
		=5292	SIZECHK					
000C		=5295+	SIZE SET 12					
		=5296+;						
		=5297+;	*****					
		=5306 ;						
		=5307	CODEBLK 15					
05E2		=5332+	ORG 1506					
		=5336	CMFMAS COMPARE MEMORY ADDRESSES					
		=5337 ;	COMPARE SMA BYTES WITH EMA BYTES TO DETERMINE RELATIVE MAGNITUDE.					
		=5338 ;	RETURNS WITH CARRY=1 IFF <SMA> >= <EMA>.					
		=5339 ;	IS CALLED AFTER ACTION HAS BEEN PERFORMED ON <SMA> TO DETERMINE IF					
		=5340 ;	TASK IS COMPLETED:					
		=5341 ;	IF CY=0 THEN <SMA> >= <EMA> ==> TERMINATE TASK.					
		=5342 ;	IF CY=1 THEN <SMA> < <EMA> ==> INC SMA AND REPEAT.					
		=5343	CMFMAS: MMOV A, SMALO					
05E2 B930		=5352+	MOV R1, #SMALO					
05E4 F1		=5353+	MOV A, @R1					
05E5 37		=5357	CPL A					
		=5358	MADD A, EMALO					
05E6 B932		=5364+	MOV R1, #EMALO					
05E8 61		=5365+	ADD A, @R1					
		=5369	MMOV A, SMAHI					
05E9 B931		=5378+	MOV R1, #SMAHI					
05EB F1		=5379+	MOV A, @R1					
05EC 37		=5383	CPL A					

5-72

LOC	OBJ	LINE	SOURCE STATEMENT	WORKSPACE	STATUS	DATE
		5411 \$	INCLUDE(<F0:KBD.MOD>)			
		=5412	CODEBLK 100			
074E		=5447+	ORG 1870			
		=5451 ;				
		=5452 ;	KEYBOARD AND DISPLAY PROCESSING ROUTINE			
		=5453 ;	CALLED PERIODICALLY WHEN KBD AND DISPLAY ARE TO BE ALIVE.			
074E D5		=5454 TIINT:	SEL RB1			
		=5455	MMOV ASAVE, A			
074F B93E		=5468+	MOV R1, #ASAVE			
0751 A1		=5469+	MOV @R1, A			
0752 23F0		=5473	MOV A, #-10H			
0754 62		=5474	MOV T, A ;RELOAD TIMER INTERVAL			
0755 27		=5475	CLR A			
0756 3E		=5476	MOVD PSEGH, A ;WRITE BLANK PATTERN TO SEG DRIVERS			
0757 3D		=5477	MOVD PSEGLO, A			
0758 FD		=5478	MOV A, CURDIG			
0759 07		=5479	DEC A			
075A 3F		=5480	MOVD PDIGIT, A ;ENERGIZE CHARACTER			
075B 0C		=5481	MOVD A, PINPUT ;LOAD ANY SWITCH CLOSURES			
075C AA		=5482	MOV ROTPAT, A			
		=5483	;WRITE NEXT SEGMENT PATTERN			
075D FD		=5484	MOV A, CURDIG			
075E 07		=5485	DEC A			
075F 0346		=5486	ADD A, #SEGMAP ;ADD CURDIG DISPLACEMENT TO BASE			
0761 A8		=5487	MOV R0, A			
0762 F0		=5488	MOV A, @R0 ;LOAD ACC W/ NEXT SEGMENT PATTERN			
0763 3D		=5489	MOVD PSEGLO, A ;ENABLE APPROPRIATE SEGMENTS			
0764 47		=5490	SWAP A			
0765 3E		=5491	MOVD PSEGH, A			
		=5492 ;				
		=5493 ;	*****			
		=5494 ;	THE NEXT CHARACTER IS NOW BEING DISPLAYED.			
		=5495 ;	THE KEYBOARD SCAN ROUTINE IS INTEGRATED INTO THE DISPLAY SCAN.			
		=5496 ;	WITH THE CURRENT ROW ENERGIZED, CHECK IF THERE ARE ANY INPUTS.			
		=5497 ;	*****			
		=5498 ;				
		=5499 ;	ROTATE BITS THROUGH THE CY WHILE INCREMENTING KEYLOC.			
		=5500 ;				
0766 B804		=5501	MOV ROTCNT, #NCOLS ;SET UP FOR <NCOLS> LOOPS THROUGH 'NXTLOC'			
		=5502 NXTLOC:	MRRC ROTPAT			
0768 FA		=5514+	MOV A, ROTPAT			
0769 67		=5518+	RRC A			
076A AA		=5529+	MOV ROTPAT, A			
076B F68B		=5532	JC SCANS ;ONE BIT IN CY INDICATES KEY NOT DOWN			
076D BE01		=5533	MOV KEYFLG, #1 ;MARK THAT AT LEAST ONE KEY WAS DETECTED			
		=5534	; \ IN THE CURRENT SCAN			
		=5535 ;				
		=5536 ;	*****			
		=5537 ;	A KEYSTROKE WAS DETECTED FOR THE CURRENT COLUMN. ITS			
		=5538 ;	POSITION IS IN REGISTER KEYLOC. SEE IF SAME KEY SENSED LAST CYCLE.			
		=5539 ;	*****			
		=5540 ;				
		=5541	MMOV A, KEYLOC			
076F B93C		=5550+	MOV R1, #KEYLOC			
0771 F1		=5551+	MOV A, @R1			

0773 DC	=5556	XRL	A, LASTKY
0774 C67C	=5557	JZ	SCAN3
	=5558 ;		
	=5559 ;	*****	
	=5560 ;	A DIFFERENT KEY WAS READ ON THIS CYCLE THAN ON THE PREVIOUS CYCLE.	
	=5561 ;	SET NREPTS TO THE DEBOUNCE PARAMETER FOR A NEW COUNTDOWN.	
	=5562 ;	*****	
	=5563 ;		
0776 B93D	=5564	MOV	R1, #NREPTS
0778 B106	=5565	MOV	@R1, #6
077A E48B	=5566	JMP	SCAN5
	=5567 ;		
	=5568 ;	*****	
	=5569 ;	SAME KEY WAS DETECTED 35 ON PREVIOUS CYCLE	
	=5570 ;	LOOK AT NREPTS: IF ALREADY ZERO, DO NOTHING.	
	=5571 ;	ELSE DECREMENT NREPTS.	
	=5572 ;	IF THIS RESULTS IN ZERO, MOVE LASTKY INTO KDDBUF.	
	=5573 ;	*****	
	=5574 ;		
	=5575 SCAN3:	MMOV	A, NREPTS
077C B93D	=5584+	MOV	R1, #NREPTS
077E F1	=5585+	MOV	A, @R1
077F C68B	=5589	JZ	SCAN5 ; IF ALREADY ZERO
0781 07	=5590	DEC	A ; INDICATE ONE MORE SUCCESSIVE KEY DETECTION
	=5591	MMOV	NREPTS, A
0782 B93D	=5604+	MOV	R1, #NREPTS
0784 A1	=5605+	MOV	@R1, A
0785 968B	=5609	JNZ	SCAN5 ; IF DECREMENT DOES NOT RESULT IN ZERO
	=5610	MMOV	KDDBUF, LASTKY ; TO MARK NEW KEY CLOSURE
0787 FC	=5633+	MOV	A, LASTKY
0788 B93B	=5639+	MOV	R1, #KDDBUF
078A A1	=5640+	MOV	@R1, A
	=5643 ;		
078B B93C	=5644 SCAN5:	MOV	R1, #KEYLOC
078D 11	=5645	INC	@R1
078E ED68	=5646	DJNZ	ROTCNT, NXTLOC
0790 EDA8	=5647	DJNZ	CURDIG, TIRET1
0792 BD08	=5648	MOV	CURDIG, #CHARNO
	=5649 ;		
	=5650 ;	*****	
	=5651 ;	THE FOLLOWING CODE SEGMENT IS USED BY THE KEYBOARD SCANNING ROUTINE.	
	=5652 ;	IT IS EXECUTED ONLY AFTER A REFRESH SEQUENCE IS COMPLETED	
	=5653 ;	*****	
	=5654 ;		
	=5655	MMOV	KEYLOC, ZERO
0794 B93C	=5666+	MOV	R1, #KEYLOC
0796 B100	=5667+	MOV	@R1, #ZERO
0798 FE	=5671	MOV	A, KEYFLG
0799 969D	=5672	JNZ	SCAN8 ; JUMP IF ANY KEYS WERE DETECTED
	=5673	MMOV	LASTKY, NEG1 ; CHANGE <LASTKY> WHEN NO KEYS ARE DOWN
079B BCFF	=5678+	MOV	LASTKY, #NEG1
079D BE00	=5682 SCAN8:	MOV	KEYFLG, #0
	=5683 ;		
	=5684 ;	*****	

LOC	OBJ	LINE	SOURCE STATEMENT	DISASSEMBLY	HEX	DEC
		=5685 ;				
		=5686 ;	KBD/DISP RETURN CODE- RESTORES SYSTEM STATUS.			
		=5687	MMOV A, RDELAY			
079F	B93F	=5696+	MOV R1, #RDELAY			
07A1	F1	=5697+	MOV A, @R1			
07A2	CGA8	=5701	JZ TIRET1			
07A4	07	=5702	DEC A			
		=5703	MMOV RDELAY, A			
07A5	B93F	=5716+	MOV R1, #RDELAY			
07A7	A1	=5717+	MOV @R1, A			
		=5721	TIRET1: MMOV A, ASAVE			
07A8	B93E	=5730+	MOV R1, #ASAVE			
07AA	F1	=5731+	MOV A, @R1			
07AB	93	=5735	RETR			
		=5736 ;				
		=5737 ;				
		=5738 ;	TOFPOL TIMER OVERFLOW POLLING SUBROUTINE.			
		=5739 ;	CALLED REPEATEDLY FROM WHEREVER KBD/DISP MUST BE ALIVE.			
		=5740 ;	MONITORS THE TIMER OVERFLOW FLAG (TOF) AND CALLS SERVICE			
		=5741 ;	ROUTINE WHEN APPROPRIATE.			
07AC	164E	=5742	TOFPOL: JTF TIINT			
07AE	83	=5743	RET			
		=5744	SIZECHK			
0061		=5747+	SIZE SET 97			
		=5748+;				
		=5749+;	*****			
		=5750	#EJECT			

LOC	OBJ	LINE	SOURCE STATEMENT	INSTRUMENT	CODE	LOC
		=5759	CODEBLK 17			
06C2		=5759+	ORG 1730			
		=5793 ;				
		=5794 ;	KBDIN: KEYBOARD INPUT SUBROUTINE.			
		=5795 ;	RETURNS ONLY AFTER A NEW KEYSTROKE HAS BEEN DETECTED AND DEBOUNCED.			
		=5796 ;	VALUE OF KEY POSITION IN SWITCH MATRIX IS			
		=5797 ;	RETURNED IN THE ACCUMULATOR.			
		=5798 ;	DISPLAY CHARACTER NOW ON BLANKED BEFORE RETURNING.			
06C2	BF03	=5799 KBDIN:	MOV XPCODE, #3			
06C4	74D1	=5800	CALL XPTST			
06C6	F4AC	=5801 KBD11:	CALL TOPPOL			
		=5802	MMOV A, KBDDBUF			
06C8	B93B	=5811+	MOV R1, #KBDDBUF			
06CA	F1	=5812+	MOV A, @R1			
06CB	F2C6	=5816	JB7 KBD11			
06CD	27	=5817	CLR A			
06CE	3E	=5818	MOVD PSEGL1, A			
06CF	3D	=5819	MOVD PSEGL0, A			
06D0	37	=5820	CPL A AND OR1			
06D1	21	=5821	XCH A, @R1			
06D2	83	=5822	RET			
		=5823	SIZECHK			
0011		=5826+	SIZE SET 17			
		=5827+;				
		=5828+;	*****			
		=5837 ;				
		=5838	CODEBLK 15			
05F1		=5863+	ORG 1521			
		=5867 ;	CLEAR: WRITES 'BLANK' CHARACTERS INTO ALL DISPLAY REGISTERS.			
		=5868 ;	RETURNS WITH NEXTPL SET TO LEFTMOST CHARACTER POSITION			
		=5869 ;	DOES NOT AFFECT ACC OR CY.			
05F1	B846	=5870 CLEAR:	MOV R0, #SEGMAP			
05F3	B908	=5871	MOV R1, #CHARNO			
05F5	B000	=5872 DBLANK:	MOV @R0, #0 ;STORE THE BLANK CODE			
05F7	18	=5873	INC R0 ;POINT TO NEXT CHARACTER TO THE LEFT			
05F8	E9F5	=5874	DJNZ R1, DBLANK			
		=5875	MMOV NEXTPL, CHARNO			
05FA	B93A	=5886+	MOV R1, #NEXTPL			
05FC	C108	=5887+	MOV @R1, #CHARNO			
05FE	83	=5891	RET			
		=5892	SIZECHK			
000E		=5895+	SIZE SET 14			
		=5896+;				
		=5897+;	*****			
		=5906 ;				
		=5907	CODEBLK 44			
06D3		=5937+	ORG 1747			
		=5941 ;	DSPACC: DISPLAY VALUE OF LOW NIBBLE OF ACC			
06D3	530F	=5942 DSPACC:	ANL A, #0FH			
06D5	03EF	=5943	ADD A, #DGPATS			
06D7	A3	=5944	MOVP A, @A			
		=5945 ;	WDISP: WRITES BIT PATTERN NOW IN ACC INTO NEXT CHARACTER POSITION			
		=5946 ;	OF THE DISPLAY (NEXTPL). INCREMENTS NEXTPL			
		=5947 ;	RESULTS IN DISPLAY BEING FILLED LEFT TO RIGHT, THEN RESTARTING			
06D8	AE	=5948 WDISP:	MOV DSPTMP, A			

LOC	OBJ	LINE	SOURCE STATEMENT
06D9	BF04	=5949	MOV XPCODE, #4
06DB	74D1	=5950	CALL XPTST
		=5951	MMOV A, NEXTPL
06DD	B93A	=5960+	MOV R1, #NEXTPL
06DF	F1	=5961+	MOV R, @R1
06E0	0345	=5965	ADD R, #SEGMAP-1
06E2	A9	=5966	MOV R1, A
06E3	FE	=5967	MOV A, DSPTMP
06E4	A1	=5968	MOV @R1, A
		=5969	MDJNZ NEXTPL, WDISP1
06E5	B93A	=5974+	MOV R1, #NEXTPL
06E7	F1	=5975+	MOV A, @R1
06E8	07	=5979+	DEC A
06E9	A1	=5984+	MOV @R1, A
06EA	96EE	=5988+	JNZ WDISP1
06EC	B108	=5990	MOV @R1, #CHARNO
06EE	83	=5991	WDISP1: RET
		=5992 ;	
		=5993 ;	DGPATS IS THE BASE FOR THE TABLE OF SEGMENT PATTERNS FOR HEX DIGITS.
		=5994 ;	HERE THE FULL HEX SET (0-F) IS INCLUDED.
		=5995 ;	
00EF		=5996	DGPATS EQU \$ AND 0FFH
		=5997 ;	
		=5998 ;	FORMAT IS PGFEDCBA IN STANDARD SEVEN-SEGMENT ENCODING CONVENTION
		=5999 ;	WHERE P REPRESENTS THE DECIMAL POINT
06EF	3F	=6000	DB 00111111B ; SEGMENT PATTERN FOR DIGIT '0'
06F0	06	=6001	DB 00000110B ; SEGMENT PATTERN FOR DIGIT '1'
06F1	5B	=6002	DB 01011011B ; SEGMENT PATTERN FOR DIGIT '2'
06F2	4F	=6003	DB 01001111B ; SEGMENT PATTERN FOR DIGIT '3'
06F3	66	=6004	DB 01100110B ; SEGMENT PATTERN FOR DIGIT '4'
06F4	6D	=6005	DB 01101101B ; SEGMENT PATTERN FOR DIGIT '5'
06F5	7D	=6006	DB 01111101B ; SEGMENT PATTERN FOR DIGIT '6'
06F6	07	=6007	DB 00000111B ; SEGMENT PATTERN FOR DIGIT '7'
06F7	7F	=6008	DB 01111111B ; SEGMENT PATTERN FOR DIGIT '8'
06F8	67	=6009	DB 01100111B ; SEGMENT PATTERN FOR DIGIT '9'
06F9	77	=6010	DB 01110111B ; SEGMENT PATTERN FOR DIGIT 'A'
06FA	7C	=6011	DB 01111100B ; SEGMENT PATTERN FOR DIGIT 'B'
06FB	39	=6012	DB 00111001B ; SEGMENT PATTERN FOR DIGIT 'C'
06FC	5E	=6013	DB 01011110B ; SEGMENT PATTERN FOR DIGIT 'D'
06FD	79	=6014	DB 01111001B ; SEGMENT PATTERN FOR DIGIT 'E'
06FE	71	=6015	DB 01110001B ; SEGMENT PATTERN FOR DIGIT 'F'
		=6016	SIZECHK
002C		=6019+	SIZE SET 44
		=6020+	
		=6021+ ;	*****
		=6030 ;	
		=6031	CODEBLK 12
04F2		=6051+	ORG 1266
		=6055 ;	DELAY SUBROUTINE WAITS FOR THE NUMBER OF COMPLETE
		=6056 ;	DISPLAY SCANS CORRESPONDING TO THE ACC CONTENTS.
		=6057 ;	USED WITH CRUDE HUMAN INTERFACES- AS WHEN OPERATOR SHOULD SEE
		=6058 ;	SOME DISPLAY CHANGE WHILE IT IS CHANGING.
		=6059	DELAY: MMOV RDELAY, A
04F2	B93F	=6072+	MOV R1, #RDELAY
04F4	A1	=6073+	MOV @R1, A

04F7 093F	=6087+	MOV	R1, #DELAY	04F7 093F	MOV	R1, #DELAY	04F7 093F	MOV	R1, #DELAY
04F9 F1	=6088+	MOV	R, @R1	04F9 F1	MOV	R, @R1	04F9 F1	MOV	R, @R1
04FA 96F5	=6092	JNZ	DELAY1	04FA 96F5	JNZ	DELAY1	04FA 96F5	JNZ	DELAY1
04FC 83	=6093	RET		04FC 83	RET		04FC 83	RET	
	=6094	SIZECHK			SIZECHK			SIZECHK	
0008	=6097+ SIZE	SET	11	0008	SIZE	SET	11	0008	SIZE
	=6098+								
	=6099+; *****								
	=6108 ;								
	=6109	CODEBLK	8		CODEBLK	8		CODEBLK	8
07AF	=6144+	ORG	1967	07AF	ORG	1967	07AF	ORG	1967
	=6148 ;	KBDPOL	POLL STATUS OF KEYBOARD INPUT ROUTINE.		KBDPOL	POLL STATUS OF KEYBOARD INPUT ROUTINE.		KBDPOL	POLL STATUS OF KEYBOARD INPUT ROUTINE.
	=6149 ;	RETURN WITH ACC BIT 7 = 0 IF KEYBOARD INPUT HAS BEEN RECEIVED.			RETURN WITH ACC BIT 7 = 0 IF KEYBOARD INPUT HAS BEEN RECEIVED.			RETURN WITH ACC BIT 7 = 0 IF KEYBOARD INPUT HAS BEEN RECEIVED.	
07AF BF05	=6150 KBDPOL:	MOV	XPCODE, #5	07AF BF05	KBDPOL:	MOV	XPCODE, #5	07AF BF05	KBDPOL:
07B1 74D1	=6151	CALL	XPTST	07B1 74D1	CALL	XPTST	07B1 74D1	CALL	XPTST
	=6152	MMOV	R, KBDBUF		MMOV	R, KBDBUF		MMOV	R, KBDBUF
07B3 B93B	=6161+	MOV	R1, #KBDBUF	07B3 B93B	MOV	R1, #KBDBUF	07B3 B93B	MOV	R1, #KBDBUF
07B5 F1	=6162+	MOV	R, @R1	07B5 F1	MOV	R, @R1	07B5 F1	MOV	R, @R1
07B6 83	=6166	RET		07B6 83	RET		07B6 83	RET	
	=6167	SIZECHK			SIZECHK			SIZECHK	
0008	=6170+ SIZE	SET	8	0008	SIZE	SET	8	0008	SIZE
	=6171+;								
	=6172+; *****								
	=6181 \$EJECT								

LOC	OBJ	LINE	SOURCE STATEMENT	SYMBOL TABLE	VALUE	TYPE
		6182 \$	INCLUDE(:F0:LINK.MOD)			
		=6183	CODEBLK 15			
07B7		=6218+	ORG 1975			
		=6222 ;EPFET	FETCH DATA BYTE FROM EP INTERNAL RAM ADDRESSED BY SMALO.			
		=6223 EPFET:	MMOV A, SMALO			
07B7 B930		=6232+	MOV R1, #SMALO			
07B9 F1		=6233+	MOV A, @R1			
07BA F4D0		=6237	CALL EPPASS			
07BC 2380		=6238	MOV A, #10000000B			
07BE F4D0		=6239	CALL EPPASS			
07C0 F4D0		=6240	CALL EPPASS			
07C2 83		=6241	RET			
		=6242	SIZECHK			
000C		=6245+ SIZE	SET 12			
		=6246+;				
		=6247+; *****				
		=6256 ;				
		=6257	CODEBLK 15			
07C3		=6292+	ORG 1987			
		=6296 ;EPSTOR	STORE DATA IN LDATA IN EP INTERNAL RAM AT <SMALO>			
07C3 FA		=6297 EPSTOR:	MOV A, LDATA			
07C4 F4D0		=6298	CALL EPPASS			
		=6299	MMOV A, SMALO			
07C6 B930		=6300+	MOV R1, #SMALO			
07C8 F1		=6309+	MOV A, @R1			
07C9 537F		=6313	ANL A, #01111111B			
07CB F4D0		=6314	CALL EPPASS			
07CD F4D0		=6315	CALL EPPASS			
07CF 83		=6316	RET			
		=6317	SIZECHK			
000D		=6320+ SIZE	SET 13			
		=6321+;				
		=6322+; *****				
		=6331 \$EJECT				

LOC	OBJ	LINE	SOURCE STATEMENT
		=6332 ;	THE FOLLOWING UTILITIES INVOLVE INTERCHANGES BETWEEN THE MP AND EP.
		=6333 ;	
		=6334	CODEBLK 11
07D0		=6369+	ORG 2000
		=6373 ;EPPASS	PASSES A SINGLE PARAMETER BYTE TO THE EP THROUGH THE LINK.
		=6374 ;	WRITE THE CONTENTS OF THE ACC TO THE LINK.
		=6375 ;	RELEASE THE EP.
		=6376 ;	READ THE LINK INTO THE ACC.
		=6377 ;	RETURN.
07D0 0A30		=6378 EPPASS:	ORL P2, #00110000B ; ENABLE LINK WRITES.
07D2 91		=6379	MOVX @R1, A ; WRITE ACC TO LINK.
07D3 99FE		=6380	ANL P1, #NOT ENBRAM ; DISABLE BREAKPOINTS.
07D5 0902		=6381	ORL P1, #ENBLNK ; SET TO BREAK ON LINK REFERENCE.
07D7 F40B		=6382	CALL EPSTEP
07D9 01		=6383	MOVX A, @R1
07DA 83		=6384	RET
		=6385	SIZECHK
0006		=6388+ SIZE	SET 11
		=6389+;	
		=6390+;	*****
		=6399 ;	
		=6400	CODEBLK 25
07DB		=6435+	ORG 2011
		=6439 ; EPSTEP	RELEASES EP TO RUN IN PRESENT MODE UNTIL AN ANTICIPATED
		=6440 ;	HARDWARE BREAK OCCURS.
		=6441 ;	(DUE TO SINGLE STEPPING, LINK OPCODE FETCH, OR LINK DATA FETCH.)
		=6442 ;	MUST OCCUR WITHIN A FINITE NUMBER OF CYCLES (<40 MP CYCLES)
		=6443 ;	OR WATCHDOG TIMER WILL ASSUME A COMMUNICATIONS ERROR
		=6444 ;	BETWEEN THE MP AND EP.
07DB F4F4		=6445 EPSTEP:	CALL EPREL
07DD 090A		=6446	MOV R1, #10
07DF 86F1		=6447 EPSTE1:	JNI EPSTE2
07E1 E9D+		=6448	DJNZ R1, EPSTE1
07E3 8910		=6449	ORL P1, #EPRSET
07E5 744F		=6450	CALL EPBRK
07E7 B88B		=6451	MOV R0, #LOW(OVLBAS+OVSZ)
07E9 746A		=6452	CALL OVLORD
07EB 99EF		=6453	ANL P1, #NOT EPRSET
07ED BA0E		=6454	MOV LDAT, #0EH
07EF 249A		=6455	JMP PEROR
07F1 744F		=6456 EPSTE2:	CALL EPBRK
07F3 83		=6457	RET
		=6458	SIZECHK
0019		=6461+ SIZE	SET 25
		=6462+;	
		=6463+;	*****
		=6472 ;	
		=6473 ;	
		=6474	\$EJECT

LOC	OBJ	LINE	SOURCE STATEMENT	TALENTA2 328002	LINE	LOC	OBJ
		=6475	CODEBLK 9				
07F4		=6510+	ORG 2036				
		=6514 ;	EPREL RELEASES EP TO RUN IN PRESENT MODE.				
		=6515 ;	SEQUENCE IS AS FOLLOWS:				
		=6516 ;	PUT MEMORY ARRAY IN EP MODE;				
		=6517 ;	RAISE /SSTEP;				
		=6518 ;	RETURN.				
07F4 99F7		=6519 EPREL:	ANL P1, #NOT CLRBF ; CLEAR BREAK F/F.				
07F6 8908		=6520	ORL P1, #CLRBFF ; RE-ENABLE BREAK F/F.				
07F8 9ABF		=6521	ANL P2, #NOT 01000000B ; ENABLE EP CONTROL OF MEM ARRAY				
07FA 8904		=6522	ORL P1, #00000100B ; FREE EP TO RUN UNTIL BREAK.				
07FC 83		=6523	RET				
		=6524	SIZECHK				
0009		=6527+ SIZE	SET 9				
		=6528+;					
		=6529+;	*****				
		=6530 ;					
		=6539 ;					
		=6540	CODEBLK 11				
034F		=6580+	ORG 847				
		=6584 ;	EPBRK REGAIN CONTROL OF MEMORY ARRAY FROM EP.				
		=6585 ;	DROP /SSTEP;				
		=6586 ;	WAIT 30 USECS. ;				
		=6587 ;	PUT MEMORY ARRAY IN MP MODE;				
		=6588 ;	RETURN.				
034F 99FB		=6589 EPBRK:	ANL P1, #NOT 00000100B ; FREEZE EMULATION PROCESSOR.				
0351 8920		=6590	ORL P1, #MODOUT ; SIGNAL EP IS NOT RUNNING USER CODE.				
0353 8905		=6591	MOV R1, #5				
0355 E955		=6592	DJNZ R1, \$; DELAY FOR EP TO FINISH INSTRUCTION.				
0357 8A40		=6593	ORL P2, #01000000B ; SEIZE CONTROL OF MEM ARRAY.				
0359 83		=6594	RET				
		=6595	SIZECHK				
000B		=6598+ SIZE	SET 11				
		=6599+;					
		=6600+;	*****				
		=6609 ;					
		=6610 ;					
		=6611	CODEBLK 16				
035A		=6651+	ORG 858				
		=6655 ;	OVSMP OVERLAY SWAP.				
		=6656 ;	SWAPS BLOCK OF DATABYTES (USER'S PROGRAM) BETWEEN MP RAM & EP PM.				
035A 8865		=6657 OVSMP:	MOV R0, #OVBUFF+OVSZ				
035C 8917		=6658	MOV R1, #OVSZ				
035E 2340		=6659	MOV A, #01000000B				
0360 3A		=6660	OUTL P2, A				
0361 C8		=6661 OVSML:	DEC R0				
0362 C9		=6662	DEC R1				
0363 81		=6663	MOVX A, @R1				
0364 20		=6664	XCH A, @R0				
0365 91		=6665	MOVX @R1, A				
0366 F9		=6666	MOV A, R1				
0367 9661		=6667	JNZ OVSML				
0369 83		=6668	RET				
		=6669	SIZECHK				
0010		=6672+ SIZE	SET 16				

```

-00(47), *****
=6683 ;
=6684 CODEBLK 14
036A =6724+ ORG 874
=6728 ;OVLORD OVERLAY LOAD.
=6729 ; MOVES BLOCK OF DATABYTES (ASSEMBLED SOURCE) FROM PG3 TO EP PM.
=6730 ; TOP OF DATA BLOCK LOADED AND BLOCK LENGTH DETERMINED BY R0 AND R1.
036A B917 =6731 OVLORD: MOV R1, #OVSZ
036C 2340 =6732 MOV A, #01000000
036E 3A =6733 OUTL F2, A
036F C8 =6734 MML01: DEC R0
0370 C9 =6735 DEC R1
0371 F8 =6736 MOV A, R0
0372 E3 =6737 MOVP3 A, R0
0373 91 =6738 MOVX @R1, A
0374 F9 =6739 MOV A, R1
0375 966F =6740 JNZ MML01
0377 83 =6741 RET
=6742 SIZECHK
000E =6745+ SIZE SET 14
=6746+;
=6747+; *****
=6756 $EJECT

```

LOC	OBJ	LINE	SOURCE STATEMENT	HEX	DEC	LOC	OBJ
		=6757 ;					
		=6758 ;	=====				
		=6759 ;					
		=6760 ;	THE REST OF THIS MODULE CONTAINS THE MINI-MONITORS WHICH OVERLAY				
		=6761 ;	THE EMULATION PROCESSOR PROGRAM RAM TO GIVE THE				
		=6762 ;	MASTER PROCESSOR ACCESS TO INTERNAL REGISTERS AND RAM OF THE EP.				
		=6763 ;	SOURCE CODE FOR MINI-MONITOR OVERLAYED OVER LOW ORDER PROGRAM RAM.				
		=6764 ;	=====				
		=6765 ;					
		=6766	DATABLK 22				
0378		=6771+	ORG 808				
		=6775 ;					
		=6776 ;	OV0- OVERLAY TO BREAK EP EXECUTION AND JUMP TO LOCATION 009H.				
		=6777 ;	LOCATION 009H REACHED WITH TOP-OF-STACK = RETURN ADDRESS+2				
		=6778 ;	DUE TO FORCED "CALL" DURING WHICH PC WAS INCREMENTED.				
		=6779 ;	LOCS 003H & 007H CALL 009H TO SIMULATE SAME CONDITION				
		=6780 ;	IF BREAK OCCURS DURING INTERRUPT CYCLE.				
		=6781 ;	SOURCE CODE FOR MINI-MONITOR OVERLAYED OVER LOW ORDER PROGRAM RAM.				
		=6782 ;					
0378		=6783 OV0BAS	EQU \$				
0378		=6784 ORG	OV0BAS				
0378 1409		=6785	CALL 009H				
037A 00		=6786	NOP				
		=6787 ;					
037B		=6788 ORG	OV0BAS+003H				
037B 1409		=6789	CALL 009H				
037D 00		=6790	NOP				
037E 00		=6791	NOP				
		=6792 ;					
037F		=6793 ORG	OV0BAS+007H				
037F 1409		=6794	CALL 009H				
0381 00		=6795	NOP				
0382 00		=6796	NOP				
0383 00		=6797	NOP				
0384 00		=6798	NOP				
0385 00		=6799	NOP				
0386 00		=6800	NOP				
0387 00		=6801	NOP				
0388 00		=6802	NOP				
0389 00		=6803	NOP				
038A 00		=6804	NOP				
038B 00		=6805	NOP				
		=6806 ;					
038C		=6807 ORG	OV0BAS+014H				
038C 0409		=6808	JMP 009H				
		=6809 ;					
		=6810	SIZECHK				
0016		=6813+ SIZE	SET 22				
		=6814+;					
		=6815+; *****					
		=6824 \$EJECT					

LOC	OBJ	LINE	SOURCE STATEMENT	OBJECT STATEMENT	LOC	OBJ
		=6825	DATABLK 22			
038E		=6830+	ORG 910			
		=6834 ;				
		=6835 ; OV3-	OVERLAY TO SAVE STATUS DATA AFTER BREAK.			
		=6836 ;	ACC, TIMER/COUNTER, PSW (WITH F1), & RAM LOC 0 PASSED SEQUENTIALLY			
		=6837 ;	TO MP.			
		=6838 ;	SOURCE CODE FOR MINI-MONITOR OVERLAYED OVER LOW ORDER PROGRAM RAM.			
		=6839 ;				
038E		=6840 OV3BAS	EQU \$			
038E		=6841 ORG	OV3BAS			
038E 0400		=6842	JMP 000H			
0390 00		=6843	NOP			
		=6844 ;				
0391		=6845 ORG	OV3BAS+003H			
0391 83		=6846	RET			
0392 00		=6847	NOP			
0393 00		=6848	NOP			
0394 00		=6849	NOP			
		=6850 ;				
0395		=6851 ORG	OV3BAS+007H			
0395 83		=6852	RET			
0396 00		=6853	NOP			
		=6854 ;				
0397		=6855 ORG	OV3BAS+009H			
0397 90		=6856	MOVX @R0, A			
0398 42		=6857	MOV A, T			
0399 90		=6858	MOVX @R0, A			
039A C7		=6859	MOV A, PSW			
039B 7611		=6860	JF1 OV3B1			
039D 53F7		=6861	ANL A, #11110111B			
0311		=6862 OV3B1	EQU \$- (LOW OV3BAS)			
039F 90		=6863	MOVX @R0, A			
03A0 C5		=6864	SEL R0			
03A1 F8		=6865	MOV A, R0			
03A2 0409		=6866	JMP 009H			
		=6867 ;				
		=6868	SIZECHK			
0016		=6871+ SIZE	SET 22			
		=6872+;				
		=6873+; *****				
		=6882 \$EJECT				

LOC	OBJ	LINE	SOURCE STATEMENT	THIRTEEN 20002	3417	100 301
		=6883	DATABLK 22	CS 1180780	1400+	
03A4		=6888+	ORG 932	100 010	1400+	1100
		=6892 ;			1400+	
		=6893 ; OV1-	OVERLAY 1 TO GIVE MP ACCESS TO EP RAM LOCS. 01H-7FH.	100 010	1400+	
		=6894 ;	SOURCE CODE FOR MINI-MONITOR OVERLAYED OVER LOW ORDER PROGRAM RAM.		1400+	
		=6895 ;			1400+	
03A4		=6896 OV1BAS	EQU \$	100 010	1400+	1100
		=6897 ;			1400+	
03A4 040A		=6898	JMP OV1B1	100 010	1400+	1100
03A6 00		=6899	NOP	100 010	1400+	1100
		=6900 ;			1400+	
03A7		=6901 ORG	OV1BAS+003H	100 010	1400+	1100
03A7 83		=6902	RET	100 010	1400+	1100
03A8 00		=6903	NOP	100 010	1400+	1100
03A9 00		=6904	NOP	100 010	1400+	1100
03AN 00		=6905	NOP	100 010	1400+	1100
		=6906 ;			1400+	
03AB		=6907 ORG	OV1BAS+007H	100 010	1400+	1100
03AB 83		=6908	RET	100 010	1400+	1100
03AC 00		=6909	NOP	100 010	1400+	1100
		=6910 ;			1400+	
03AD		=6911 ORG	OV1BAS+009H	100 010	1400+	1100
03AD 90		=6912	MOVX @R0, A	100 010	1400+	1100
		=6913 ;			1400+	
000A		=6914 OV1B1	EQU \$-OV1BAS	100 010	1400+	1100
		=6915 ;			1400+	
03AE 80		=6916	MOVX A, @R0	100 010	1400+	1100
03AF A8		=6917	MOV R0, A	100 010	1400+	1100
03B0 80		=6918	MOVX A, @R0	100 010	1400+	1100
03B1 F213		=6919	JB7 OV1B2	100 010	1400+	1100
03B3 28		=6920	XCH A, R0	100 010	1400+	1100
03B4 A0		=6921	MOV @R0, A	100 010	1400+	1100
03B5 0409		=6922	JMP 009H	100 010	1400+	1100
		=6923 ;			1400+	
0313		=6924 OV1B2	EQU \$-LOW OV1BAS	100 010	1400+	1100
		=6925 ;			1400+	
03B7 F0		=6926	MOV A, @R0	100 010	1400+	1100
03B8 0409		=6927	JMP 009H	100 010	1400+	1100
		=6928 ;			1400+	
		=6929	SIZECHK	100 010	1400+	1100
0016		=6932+ SIZE	SET 22	100 010	1400+	1100
		=6933+;			1400+	
		=6934+; *****			1400+	
		=6943 \$EJECT			1400+	

03DA	=6949+	ORG	954				
	=6953 ;						
	=6954 ; OV2-	OVERLAY TO RESTORE EP STATUS SAVED ON BREAK AND RESUME USER'S PROGRAM.					
	=6955 ;	SOURCE CODE FOR MINI-MONITOR OVERLAYED OVER LOW ORDER PROGRAM RAM.					
	=6956 ;						
03BA	=6957 OV2BAS	EQU	\$				
03BA	=6958 ORG	OV2BAS					
03DA 0400	=6959	JMP	000H				
03BC 00	=6960	NOP					
	=6961 ;						
03BD	=6962 ORG	OV2BAS+003H					
03BD 83	=6963	RET					
03BE 00	=6964	NOP					
03BF 00	=6965	NOP					
03C0 00	=6966	NOP					
	=6967 ;						
03C1	=6968 ORG	OV2BAS+007H					
03C1 83	=6969	RET					
03C2 00	=6970	NOP					
	=6971 ;						
03C3	=6972 ORG	OV2BAS+009H					
03C3 90	=6973	MOVX	0R0, A				
	=6974 ;						
03C4 80	=6975	MOVX	A, 0R0				
03C5 A8	=6976	MOV	R0, A				
03C6 80	=6977	MOVX	A, 0R0				
03C7 D7	=6978	MOV	PSW, A				
03C8 A5	=6979	CLR	F1				
03C9 B5	=6980	CPL	F1				
03CA 7213	=6981	JB3	OV2B1				
03CC A5	=6982	CLR	F1				
	=6983 ;						
0313	=6984 OV2B1	EQU	\$-LOW OV2BAS				
	=6985 ;						
03CD 80	=6986	MOVX	A, 0R0				
03CE 62	=6987	MOV	T, A				
03CF 80	=6988	MOVX	A, 0R0				
03D0 93	=6989	RETR					
	=6990	SIZECHK					
0017	=6993+ SIZE	SET	23				
	=6994+;						
	=6995+; *****						
	=7004 \$EJECT						

LOC	OBJ	LINE	SOURCE STATEMENT	TRANSLATED STATEMENT	LOC	OBJ
		7005 ;				
		7006	CODEBLK 11			
03D1		7046+	ORG 977			
03D1 0A80		7050	XPTST: ORL P2, #00H			
03D3 0A		7051	IN A, P2			
03D4 9A7F		7052	ANL P2, #(NOT 00H)			
03D6 F2D9		7053	JB7 \$+3			
03D8 83		7054	RET			
03D9 F5		7055	SEL MB1			
03DA 0400		7056	JMP 000H			
		7057	SIZECHK			
000B		7060+	SIZE SET 11			
		7061+;				
		7062+;	*****			
		7071 ;				
		7072	CODEBLK 13			
03DC		7112+	ORG 988			
03DC 28432931		7116	DB '(C)1979 INTEL'			
03E0 39373920						
03E4 494E5445						
03E8 4C						
		7117	SIZECHK			
000D		7120+	SIZE SET 13			
		7121+;				
		7122+;	*****			
		7131 ;				
		7132 ;				
		7133	RSOURCE			
0100		7135+	PGSIZE SET ORGPG0-000H ; BYTES USED ON PAGE 0			
00FD		7136+	PGSIZE SET ORGPG1-100H ; BYTES USED ON PAGE 1			
0100		7137+	PGSIZE SET ORGPG2-200H ; BYTES USED ON PAGE 2			
00E9		7138+	PGSIZE SET ORGPG3-300H ; BYTES USED ON PAGE 3			
00FD		7139+	PGSIZE SET ORGPG4-400H ; BYTES USED ON PAGE 4			
00FF		7140+	PGSIZE SET ORGPG5-500H ; BYTES USED ON PAGE 5			
00FF		7141+	PGSIZE SET ORGPG6-600H ; BYTES USED ON PAGE 6			
00FD		7142+	PGSIZE SET ORGPG7-700H ; BYTES USED ON PAGE 7			
		7143+JEJECT				

LOC	OBJ	LINE	SOURCE STATEMENT	ADDRESS	HEX	ASCII
		7145 ;	*****			
		7146 ;				
		7147 ;	FILL ALL UNUSED MEMORY LOCATIONS WITH NOP OPCODES			
		7148 ;				
		7149 ;	*****			
		7150 ;				
		7151 \$GEN				
		7158 ;				
01FD		7160	ORG ORGPG1			
		7161	REPT (200H - ORGPG1)			
		7162	DB 0			
		7163	ENDM			
01FD 00		7164+	DB 0			
01FE 00		7165+	DB 0			
01FF 00		7166+	DB 0			
		7168 ;				
		7175 ;				
03E9		7177	ORG ORGPG3			
		7178	REPT (400H - ORGPG3)			
		7179	DB 0			
		7180	ENDM			
03E9 00		7181+	DB 0			
03EA 00		7182+	DB 0			
03EB 00		7183+	DB 0			
03EC 00		7184+	DB 0			
03ED 00		7185+	DB 0			
03EE 00		7186+	DB 0			
03EF 00		7187+	DB 0			
03F0 00		7188+	DB 0			
03F1 00		7189+	DB 0			
03F2 00		7190+	DB 0			
03F3 00		7191+	DB 0			
03F4 00		7192+	DB 0			
03F5 00		7193+	DB 0			
03F6 00		7194+	DB 0			
03F7 00		7195+	DB 0			
03F8 00		7196+	DB 0			
03F9 00		7197+	DB 0			
03FA 00		7198+	DB 0			
03FB 00		7199+	DB 0			
03FC 00		7200+	DB 0			
03FD 00		7201+	DB 0			
03FE 00		7202+	DB 0			
03FF 00		7203+	DB 0			
		7205 ;				
04FD		7207	ORG ORGPG4			
		7208	REPT (500H - ORGPG4)			
		7209	DB 0			
		7210	ENDM			
04FD 00		7211+	DB 0			
04FE 00		7212+	DB 0			
04FF 00		7213+	DB 0			
		7215 ;				
05FF		7217	ORG ORGPG5			
		7218	REPT (600H - ORGPG5)			

© 1996 Intel Corporation

All mnemonics copyrighted © Intel Corporation 1976.

All mnemonics copyrighted © Intel Corporation 1976.

5-92

CROSS REFERENCE COMPLETE

All mnemonics copyrighted © Intel Corporation 1976.

DEBNCE	630#																
DECLAR	170#	584	600	616	632	648	670	685	700	715	739	756	773	790	807	824	
	848	869	890	911	932	964	973	982	991	1000	1009	1018	1027	1036	1045	1054	
	1063	1072	1081	1090	1099	1108	1117	1126	1135	1144	1153	1162	1171	1180	1189	1198	
	1207	1216	1225	1234	1243	1252	1261	1270	1279	1288	1297	1306	1315	1324	1333	1342	
DECSM1	5286	5291#															
DECSMA	2630	5282#															
DELAY	3105	6059#															
DELAY1	6077#	6092															
DERROR	2033	2063#															
DFILL	2040	2096#															
DGO	2039	2094#															
DGPATS	5943	5996#															
DGR	2046	2100#															
DINTRG	2051	2124#															
DLST	2041	2098#															
DMOD	2038	2092#															
DNOBRK	2055	2129#															
DONE	3419	3595#															
DPA	2058	2135#															
DPRBRK	2052	2120#															
DFRMEM	2048	2114#															
DREC	2042	2100#															
DREL	2043	2102#															
DRM	2050	2118#															
DRUN	2035	2078#															
DSB	2044	2104#															
DSGNON	2034	2070#															
DSPACC	2276	2279	2281	2328	2564	2566	5942#										
DSPHI	2268	2276#															
DSPLO	2275	2280#															
DSPML	2273	2279#															
DSPMID	2277#	3375															
DSPTIM	1041#	3100															
DSPIMP	820#	5948	5967														
DSS	2057	2133#															
DTR	2059	2137#															
DWBRK	2056	2131#															
ELSIF1	2186	2188	2202#														
ELSIF2	2204	2207	2213#														
EMAH1	1140#	5390															
EMALO	1131#	5364															
ENBLNK	529#	3197	6381														
ENBRAM	528#	3197	6390														
ENDF1	3888#	3893															
ENDFIL	3872	3884#															
ENDREC	4064#																
EOFREC	3887	3901#															
EPACC	969#	2977	3219	3374	4884	5104											
EPBRK	1425	3183	6450	6456	6589#												
EPCNT	2921#	3107	3125														
EPCON1	2785	2805#															
EPCONT	2728	2783#															
EPFET	3319	3343	4852	6223#													
EPPASS	2937	2952	2967	2982	3205	3224	3243	3262	6237	6239	6240	6298	6314	6315	6378#		
EPPCHI	1014#	2779	2914	3362	3370												
EPPCLO	1005#	2752	2821	3335													

5-96

INCH	5239#																
INCM1	5241	5246#															
INIT	1409#																
INITLP	1410#	1423															
INPAD1	2187#	2198															
INPADR	1835	2170#	2490														
INPKEY	1543	1675	1828	1846	2196	2345	2463	2500	2650#	3115	3119						
INVALS	1346#	1381	1417														
ITMP	786#	1557	1590	1595	1597	1625	1626	1646	1647	1705	1712	1715	1725	1732	1761	1830	
	1832	1833	1837														
JGORES	2408	2429#															
JMPTBL	2385	2399#															
JIOFIL	2402	2426#															
JTOGO	2401	2424#															
JIOIST	2403	2419#															
JTOMOD	2400	2410#															
JTOREC	2404	2412#															
JTOREL	2405	2416#															
KBD8UF	1212#	2334	2340	5639	5811	6161											
KBDI1	5801#	5816															
KBDIN	2658	5799#															
KBDPOL	3047	3106	6150#														
KCLRB	1515#	1908															
KEY	769#	1544	1593	1740	1843	2187	2202	2205	2322	2346	2460	2590	2597	2613	2622	2627	
	2659	2783	3116	3120													
KEYCLR	1505#	2323	2628														
KEYDM	1504#	1923	1926														
KEYEND	1501#	1545	1844	2206	2347	2461	2618	3117									
KEYFIL	1499#	1903															
KEYFLG	949#	5533	5671	5682													
KEYGO	1512#	1902															
KEYLOC	1221#	5550	5644	5660	5666												
KEYLST	1510#	1904															
KEYMOD	1513#	1901															
KEYNKT	1500#	2203	2623	2784	3121												
KEYPAT	1503#	1929															
KEYPM	1508#	1923	1926														
KEYREC	1506#	1905															
KEYREG	1509#	1923															
KEYREL	1502#	1906															
KEYTRA	1507#	1929															
KGORES	1511#	1909															
KSETB	1514#	1907															
LASTKY	907#	5555	5556	5626	5633	5670											
LDATR	752#	1858	2211	2317	2327	2432	2436	2562	2565	2607	2614	2632	2628	2835	2919	3088	
	3321	3344	3346	3367	3368	3526	3598	3629	3641	3648	3660	3666	3715	3868	4154	4157	
	4160	4662	4669	4705	5028	5033	5071	5078	5106	5112	6297	6454					
LDBYTE	3867#	3876															
LFEBR1	4897	4899#															
LFEBRK	4780	4781	4890#														
LFEDM	4777	4797	4813	4832#													
LFEINT	4779	4870#															
LFEPM	4776	4783#															
LFER0	4851	4854#															
LFEREG	4778	4836#															
LFETBL	4773	4776#															
LFETCH	2561	3867	4704#														

STRUTL 0019	STSAVE 0500	TCRLFO 0102	TIINT 074E	TIRET1 07F8	TOFPOL 07AC	TYOUT 0040	TYPE 0037
UPAD1 017C	UPADR 0178	VER5NO 0029	WBRK 0016	WDISP 06D8	WDISP1 06EE	XPCODE 0007	XPIEST 03D1
ZERO 0000							

ASSEMBLY COMPLETE, NO ERRORS

0000	0001	0002	0003	0004	0005	0006	0007	0008	0009	0010	0011	0012	0013	0014	0015	0016	0017	0018	0019	0020	0021	0022	0023	0024	0025	0026	0027	0028	0029	0030	0031	0032	0033	0034	0035	0036	0037	0038	0039	0040	0041	0042	0043	0044	0045	0046	0047	0048	0049	0050	0051	0052	0053	0054	0055	0056	0057	0058	0059	0060	0061	0062	0063	0064	0065	0066	0067	0068	0069	0070	0071	0072	0073	0074	0075	0076	0077	0078	0079	0080	0081	0082	0083	0084	0085	0086	0087	0088	0089	0090	0091	0092	0093	0094	0095	0096	0097	0098	0099	0100	0101	0102	0103	0104	0105	0106	0107	0108	0109	0110	0111	0112	0113	0114	0115	0116	0117	0118	0119	0120	0121	0122	0123	0124	0125	0126	0127	0128	0129	0130	0131	0132	0133	0134	0135	0136	0137	0138	0139	0140	0141	0142	0143	0144	0145	0146	0147	0148	0149	0150	0151	0152	0153	0154	0155	0156	0157	0158	0159	0160	0161	0162	0163	0164	0165	0166	0167	0168	0169	0170	0171	0172	0173	0174	0175	0176	0177	0178	0179	0180	0181	0182	0183	0184	0185	0186	0187	0188	0189	0190	0191	0192	0193	0194	0195	0196	0197	0198	0199	0200	0201	0202	0203	0204	0205	0206	0207	0208	0209	0210	0211	0212	0213	0214	0215	0216	0217	0218	0219	0220	0221	0222	0223	0224	0225	0226	0227	0228	0229	0230	0231	0232	0233	0234	0235	0236	0237	0238	0239	0240	0241	0242	0243	0244	0245	0246	0247	0248	0249	0250	0251	0252	0253	0254	0255	0256	0257	0258	0259	0260	0261	0262	0263	0264	0265	0266	0267	0268	0269	0270	0271	0272	0273	0274	0275	0276	0277	0278	0279	0280	0281	0282	0283	0284	0285	0286	0287	0288	0289	0290	0291	0292	0293	0294	0295	0296	0297	0298	0299	0300	0301	0302	0303	0304	0305	0306	0307	0308	0309	0310	0311	0312	0313	0314	0315	0316	0317	0318	0319	0320	0321	0322	0323	0324	0325	0326	0327	0328	0329	0330	0331	0332	0333	0334	0335	0336	0337	0338	0339	0340	0341	0342	0343	0344	0345	0346	0347	0348	0349	0350	0351	0352	0353	0354	0355	0356	0357	0358	0359	0360	0361	0362	0363	0364	0365	0366	0367	0368	0369	0370	0371	0372	0373	0374	0375	0376	0377	0378	0379	0380	0381	0382	0383	0384	0385	0386	0387	0388	0389	0390	0391	0392	0393	0394	0395	0396	0397	0398	0399	0400	0401	0402	0403	0404	0405	0406	0407	0408	0409	0410	0411	0412	0413	0414	0415	0416	0417	0418	0419	0420	0421	0422	0423	0424	0425	0426	0427	0428	0429	0430	0431	0432	0433	0434	0435	0436	0437	0438	0439	0440	0441	0442	0443	0444	0445	0446	0447	0448	0449	0450	0451	0452	0453	0454	0455	0456	0457	0458	0459	0460	0461	0462	0463	0464	0465	0466	0467	0468	0469	0470	0471	0472	0473	0474	0475	0476	0477	0478	0479	0480	0481	0482	0483	0484	0485	0486	0487	0488	0489	0490	0491	0492	0493	0494	0495	0496	0497	0498	0499	0500	0501	0502	0503	0504	0505	0506	0507	0508	0509	0510	0511	0512	0513	0514	0515	0516	0517	0518	0519	0520	0521	0522	0523	0524	0525	0526	0527	0528	0529	0530	0531	0532	0533	0534	0535	0536	0537	0538	0539	0540	0541	0542	0543	0544	0545	0546	0547	0548	0549	0550	0551	0552	0553	0554	0555	0556	0557	0558	0559	0560	0561	0562	0563	0564	0565	0566	0567	0568	0569	0570	0571	0572	0573	0574	0575	0576	0577	0578	0579	0580	0581	0582	0583	0584	0585	0586	0587	0588	0589	0590	0591	0592	0593	0594	0595	0596	0597	0598	0599	0600	0601	0602	0603	0604	0605	0606	0607	0608	0609	0610	0611	0612	0613	0614	0615	0616	0617	0618	0619	0620	0621	0622	0623	0624	0625	0626	0627	0628	0629	0630	0631	0632	0633	0634	0635	0636	0637	0638	0639	0640	0641	0642	0643	0644	0645	0646	0647	0648	0649	0650	0651	0652	0653	0654	0655	0656	0657	0658	0659	0660	0661	0662	0663	0664	0665	0666	0667	0668	0669	0670	0671	0672	0673	0674	0675	0676	0677	0678	0679	0680	0681	0682	0683	0684	0685	0686	0687	0688	0689	0690	0691	0692	0693	0694	0695	0696	0697	0698	0699	0700	0701	0702	0703	0704	0705	0706	0707	0708	0709	0710	0711	0712	0713	0714	0715	0716	0717	0718	0719	0720	0721	0722	0723	0724	0725	0726	0727	0728	0729	0730	0731	0732	0733	0734	0735	0736	0737	0738	0739	0740	0741	0742	0743	0744	0745	0746	0747	0748	0749	0750	0751	0752	0753	0754	0755	0756	0757	0758	0759	0760	0761	0762	0763	0764	0765	0766	0767	0768	0769	0770	0771	0772	0773	0774	0775	0776	0777	0778	0779	0780	0781	0782	0783	0784	0785	0786	0787	0788	0789	0790	0791	0792	0793	0794	0795	0796	0797	0798	0799	0800	0801	0802	0803	0804	0805	0806	0807	0808	0809	0810	0811	0812	0813	0814	0815	0816	0817	0818	0819	0820	0821	0822	0823	0824	0825	0826	0827	0828	0829	0830	0831	0832	0833	0834	0835	0836	0837	0838	0839	0840	0841	0842	0843	0844	0845	0846	0847	0848	0849	0850	0851	0852	0853	0854	0855	0856	0857	0858	0859	0860	0861	0862	0863	0864	0865	0866	0867	0868	0869	0870	0871	0872	0873	0874	0875	0876	0877	0878	0879	0880	0881	0882	0883	0884	0885	0886	0887	0888	0889	0890	0891	0892	0893	0894	0895	0896	0897	0898	0899	0900	0901	0902	0903	0904	0905	0906	0907	0908	0909	0910	0911	0912	0913	0914	0915	0916	0917	0918	0919	0920	0921	0922	0923	0924	0925	0926	0927	0928	0929	0930	0931	0932	0933	0934	0935	0936	0937	0938	0939	0940	0941	0942	0943	0944	0945	0946	0947	0948	0949	0950	0951	0952	0953	0954	0955	0956	0957	0958	0959	0960	0961	0962	0963	0964	0965	0966	0967	0968	0969	0970	0971	0972	0973	0974	0975	0976	0977	0978	0979	0980	0981	0982	0983	0984	0985	0986	0987	0988	0989	0990	0991	0992	0993	0994	0995	0996	0997	0998	0999
------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------

?A	105#	1614	1629	1637	1650	1658	1721	1787	1806	1818	1977	1985	1999	2177	2388	2444
	2546	2586	2719	2788	2796	2843	2857	2865	2898	2910	2928	2943	2958	2973	3007	3068
	3096	3207	3215	3226	3234	3245	3253	3264	3272	3288	3302	3310	3323	3331	3350	3358
	3434	3442	3453	3461	3472	3480	3491	3499	3515	3562	3583	3637	3958	3966	3986	4001
	4016	4070	4393	4401	4592	4764	4788	4803	4819	4841	4859	4875	4962	4986	5001	5017
	5042	5095	5167	5180	5196	5348	5360	5374	5386	5456	5464	5546	5580	5592	5600	5692
	5704	5712	5726	5807	5956	6060	6068	6083	6157	6228	6304					
?ASAVE	1235#	5460	5466	5722	5728											
?B	1280#	4413	4413	4413	4419	4459	4469	4569	4579							
?B0PNT	117#	746	754#	763	771#	780	788#	797	805#	814	822#	831	839#			
?B0R2	110#	754														
?B0R3	111#	771														
?B0R4	112#	788														
?B0R5	113#	805														
?B0R6	114#	822														
?B0R7	115#	839														
?B1PNT	126#	859	867#	880	888#	901	909#	922	930#	943	951#					
?B1R2	119#	867														
?B1R3	120#	888														
?B1R4	121#	909														
?B1R5	122#	930														
?B1R6	123#	951														
?B1R7	124#															
?BCODE	1163#	1561	1561	1561	1567	1610	1616	2390								
?BINOP	415#	1817	2387	2439	2909	3632	5179	5359	5385							
?BIT50	4217#	4411														
?BUFCN	1262#	3438	3444	3511	3517	3532	3542	3962	3968	3982	3988	4044	4054			
?BUFL	649#															
?CHARN	585#	5876														
?CHKSU	791#	3426	3426	3426	3558	3564	3571	3571	3858	3858	3858	4066	4072	4079	4079	
?CONST	104#	585	586	590	594	601	602	606	610	617	618	622	626	633	634	638
	642	649	650	654	658	671	672	676	680	686	687	691	695	701	702	706
	710	716	717	721	725	4217	4218	4222	4226							
?CURDI	912#															
?DEBNC	617#															
?DSPTI	1037#	3092	3098													
?DSPTM	808#															
?EMNHI	1136#	5388														
?EMNLO	1127#	5362														
?EPNCC	965#	2969	2975	3211	3217											
?EPNCH	1010#	2761	2777	2912	3354	3360										
?EPPCL	1001#	2734	2750	2806	2814	2819	3327	3333								
?EPPSW	974#	2792	2798	2839	2845	2894	2900	2939	2945	3249	3255	3284	3290			
?EPR0	992#	2924	2930	3268	3274	4855	4861	5060	5082							
?EPTIM	983#	2954	2960	3230	3236											
?FORM1	295#	1615	1634	1655	1688	1695	1722	1745	1788	1807	1819	1982	2000	2013	2178	2389
	2441	2445	2547	2587	2720	2735	2742	2762	2769	2793	2811	2818	2844	2862	2877	2899
	2911	2929	2944	2959	2974	3008	3069	3097	3212	3231	3250	3269	3289	3307	3328	3355
	3439	3458	3477	3496	3516	3531	3563	3584	3634	3638	3807	3814	3834	3841	3963	3987
	4002	4017	4043	4071	4263	4270	4290	4297	4322	4396	4439	4458	4550	4568	4593	4645
	4652	4765	4789	4804	4820	4842	4860	4876	4963	4987	5002	5018	5043	5061	5068	5096
	5168	5181	5197	5349	5361	5375	5387	5461	5504	5547	5581	5597	5616	5623	5693	5709
	5727	5808	5957	5971	6065	6084	6158	6229	6305							
?FORM2	319#	1638	1659	1692	1702	1986	2739	2749	2766	2776	2797	2815	2825	2866	3216	3235
	3254	3273	3311	3332	3359	3443	3462	3481	3500	3811	3821	3838	3848	3967	4267	4277
	4294	4304	4402	4649	4659	5065	5081	5465	5601	5620	5636	5713	6069			
?FORM3	339#	1755	2023	2452	2887	3541	3651	4053	4332	4449	4468	4560	4578	5520	5981	

All mnemonics copyrighted © Intel Corporation 1976.

?R1	99#	4289	4305	4312	4312											
?RAM	103#	965	966	974	975	983	984	992	993	1001	1002	1010	1011	1019	1020	1028
	1029	1037	1038	1046	1047	1055	1056	1064	1065	1073	1074	1082	1083	1091	1092	1100
	1101	1109	1110	1118	1119	1127	1128	1136	1137	1145	1146	1154	1155	1163	1164	1172
	1173	1181	1182	1190	1191	1199	1200	1208	1209	1217	1218	1226	1227	1235	1236	1244
	1245	1253	1254	1262	1263	1271	1272	1280	1281	1289	1290	1298	1299			
?R80	101#	740	741	745	757	758	762	774	775	779	791	792	796	808	809	813
	825	826	830													
?R81	102#	849	850	854	858	870	871	875	879	891	892	896	900	912	913	917
	921	933	934	938	942											
?RELA	1244#	5688	5694	5708	5714	6064	6070	6079	6085							
?RECTY	1271#	3495	3501	3579	3585											
?REGC	1289#	4397	4403	4440	4450	4551	4561	4508	4594							
?ROTCN	870#															
?ROTPA	849#	5505	5512	5512	5521	5527	5527									
?RSAYE	144#	591	595	607	611	623	627	639	643	655	659	677	681	692	696	707
	711	722	726	746	763	780	797	814	831	855	859	876	880	897	901	918
	922	939	943	4223	4227											
?SEGMA	1308#															
?SIZE	255#	1303	1437	1861	1931	2141	2218	2284	2351	2502	2635	2662	3132	3378	3601	3669
	3718	3753	3904	4090	4125	4164	4201	4343	4479	4603	4679	4708	4901	5122	5208	5240
	5293	5397	5745	5824	5893	6017	6095	6168	6243	6318	6386	6459	6525	6596	6670	6743
	6811	6869	6930	6991	7058	7118										
?SMHI	1118#	2405	2485	2485	2491	2757	2765	2770	3457	3463	3802	3810	3815	4784	4790	4982
	4988	5182	5370	5376												
?SMALO	1109#	2730	2730	2743	2861	2867	2878	2888	3306	3312	3476	3482	3829	3837	3842	4799
	4805	4815	4821	4837	4843	4871	4877	4997	5003	5013	5019	5038	5044	5091	5097	5192
	5198	5344	5350	6224	6230	6300	6306									
?START	1339#	1383	1383	1391	1405#	1437	1437	1445	1533#	1861	1861	1869	1882#	1931	1931	1939
	1957#	2141	2141	2149	2162#	2218	2218	2226	2244#	2204	2284	2292	2310#	2351	2351	2359
	2382#	2502	2502	2510	2533#	2635	2635	2643	2656#	2662	2662	2670	2699#	3132	3132	3140
	3173#	3378	3378	3386	3414#	3601	3601	3609	3622#	3669	3669	3677	3695#	3718	3718	3726
	3745#	3753	3753	3761	3796#	3904	3904	3912	3951#	4090	4090	4098	4116#	4125	4125	4133
	4151#	4164	4164	4172	4190#	4201	4201	4209	4254#	4343	4343	4351	4385#	4479	4479	4487
	4525#	4603	4603	4611	4635#	4679	4679	4687	4700#	4708	4708	4716	4754#	4901	4901	4909
	4952#	5122	5122	5130	5150#	5208	5208	5216	5235#	5248	5248	5256	5279#	5293	5293	5301
	5334#	5397	5397	5405	5449#	5745	5745	5753	5791#	5824	5824	5832	5865#	5893	5893	5901
	5939#	6017	6017	6025	6053#	6095	6095	6103	6146#	6168	6168	6176	6220#	6243	6243	6251
	6294#	6318	6318	6326	6371#	6386	6386	6394	6437#	6459	6459	6467	6512#	6525	6525	6533
	6582#	6596	6596	6604	6653#	6670	6670	6678	6726#	6743	6743	6751	6773#	6811	6811	6819
	6832#	6869	6869	6877	6890#	6930	6930	6938	6951#	6991	6991	6999	7048#	7058	7058	7066
	7114#	7118	7118	7126												
?STRTH	1253#	1981	1987	1995	2001	2014	2024									
?TYPE	1172#	1577	1577	1577	1583	1746	1756	1769	1769	1769	1775	1820	2440	2446	2453	2542
	2548	3003	3009	3064	3070	4760	4766	4958	4964	5163	5169					
?UNARY	459#	1744	2012	2876	3530	4042	4321	4430	4457	4549	4567	5503	5970			
?VERSN	1046#															
?XPCOD	825#															
?ZERO	671#	1559	1575	1767	2483	3424	3856	5636								
AFETCH	4704	4759#														
ASAVE	1239#	5468	5730													
ASCERR	3704	3711	3715#													
B	1284#	4415	4421	4461	4529	4571										
BCODE	1167#	1563	1569	1598	1618	2392										
BIT50	4230#	4422														
BRKEND	2462	2500#														
BRKERR	3000	3080#														

All mnemonics copyrighted © Intel Corporation 1976.

C and D

Appendix 1

APPENDIX C COMMAND SUMMARY

The following is a summary of the commands implemented by the HSE-49 emulator monitor. Within each command group, tokens in each column indicate options the user has when invoking those commands.

Tokens in square brackets indicate dedicated keys on the keyboard (some keys having shared functions); angle brackets enclose hex digit strings used to specify an address or data parameter. Parameters in parentheses are optional, with the effects explained above. The notation used is as follows:

<SMA> — Starting Memory Address for block command,
<EMA> — Ending Memory Address for block command,
<LOC> — LOcation for individual accesses,
<DATA> — DATA byte.

Asterisks (*) indicate the default condition for each command; thus that token is optional and serves to regularize the command syntax.

Program/data entry and verification commands:

```
[EXAM] [PROG MEM]* <LOC> [,] [NEXT]
        [DATA MEM] [PREV]
        [REGISTER] [,]
        [HWRE REG]
        [PROG BRK]
        [DATA BRK]
```

Program/data initialization commands:

```
[FILL] [PROG MEM]* <SMA> [,] <EMA> [,] <DATA> [,]
        [DATA MEM]
        [REGISTER]
        [HWRE REG]
        [PROG BRK]
        [DATA BRK]
```

Intellec® development system or TTY interface commands (for transferring HEX format files):

```
[UPLOAD] [PROG MEM]* <SMA> [,] <EMA> [,]
          [DATA MEM]
          [REGISTER]
          [HWRE REG]
          [PROG BRK]
          [DATA BRK]

[DNLOAD] [PROG MEM]* [,]
          [DATA MEM]
          [REGISTER]
          [HWRE REG]
          [PROG BRK]
          [DATA BRK]
```

Formatted data dump to TTY or CRT:

```
[LIST] [PROG MEM]* <SMA> [,] <EMA> [,]
        [DATA MEM]
        [REGISTER]
        [HWRE REG]
        [PROG BRK]
        [DATA BRK]
```

Program execution commands:

```
[GO] [NO BREAK]* <SMA> [,]
      [W/ BREAK] [,]
      [SING STP]
      [AUTO BRK]
      [AUTO STP]

[GO/RST] [NO BREAK]* [,]
          [W/ BREAK]
          [SING STP]
          [AUTO BRK]
          [AUTO STP]
```

Breakpoint setting and clearing:

```
[SET BRK] [PROG MEM]* <LOC> [,] <LOC> ... [,]
          [DATA MEM]

[CLR BRK] [PROG MEM]* <LOC> [,] <LOC> ... [,]
          [DATA MEM]
```

APPENDIX D ERROR MESSAGES

The following error message codes are used by the monitor software to report an operator or hardware error. Errors may be cleared by pressing [CLR/PREV] or [END/]. The format used for reporting errors is "Error - .n" where "n" is a hex digit.

Operator Errors

1. Illegal command initiator.
2. Illegal command modifier or parameter digit.
3. Illegal terminator for Examine command.
4. Illegal attempt to clear Error mode.
- 5-9. Not used.

Hardware Errors

- A. ASCII error — non-hex digit encountered in data field of hex format record.
- B. Breakpoint error. Break logic activated though breakpoints not enabled.
- C. Hex format record checksum error. Note — the checksum will not be verified if the first character of the checksum field is a question mark ("?",) rather than a hexadecimal digit. This allows object files to be patched using the ISIS text editor without the necessity of manually recomputing the checksum value.
- D. Not used.
- E. Execution processor failed to respond to a command or parameter passed to it by the master processor. EP automatically reset. EP internal status may be lost. Program memory not affected.
- F. Not used.

June 1978

Microcontroller includes a-d converter for lowest-cost analog interfacing

by W. Check, E. Cheng, G. Hill, M. Holler, and J. Miller
Electronics/ May 25, 1978

Microcontroller includes a-d converter for lowest-cost analog interfacing

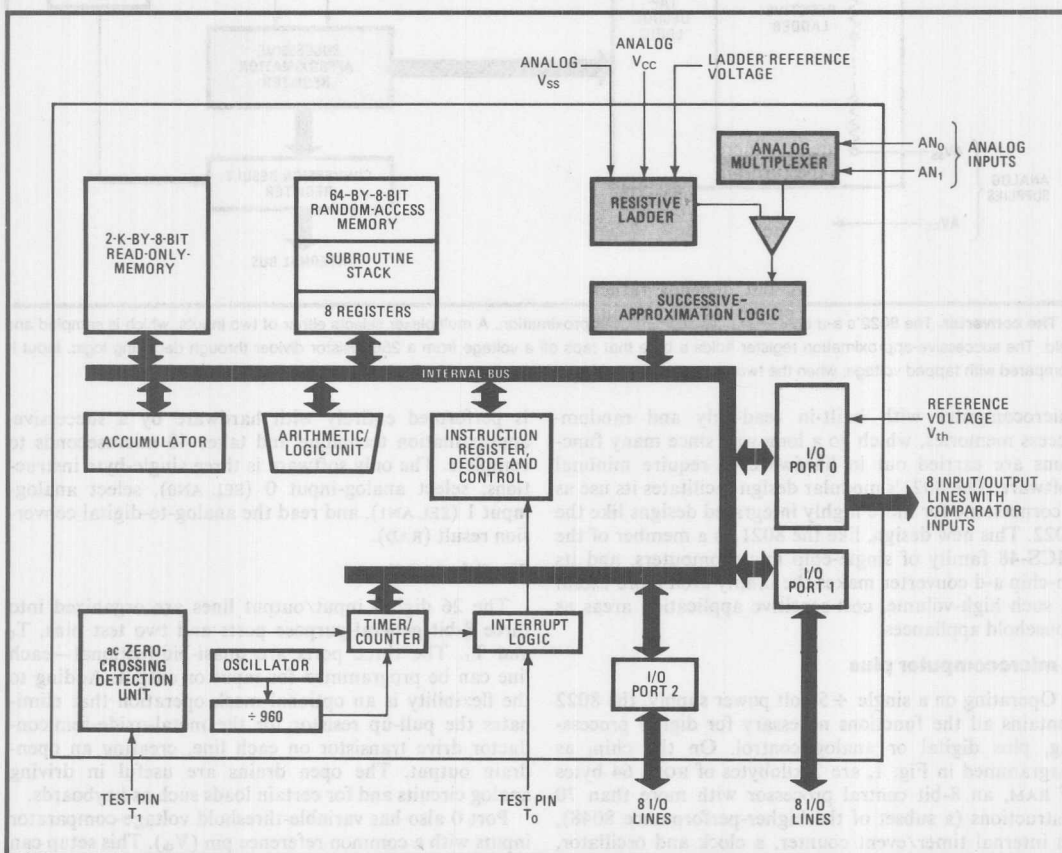
Adding hardware for analog-to-digital conversion to a single-chip microcomputer cuts interface software and component count for high-volume control applications

by W. Check, E. Cheng, G. Hill, M. Hollen, and J. Miller, Intel Corp., Santa Clara, Calif.

Microcomputers' plunging size and cost are creating a rising new market: low-cost controllers that end up in automobiles, appliances, and consumer products. Now that the technology is available to integrate a high-performance 8-bit analog-to-digital converter and a microcomputer on a single chip, the tremendous need for

low-cost analog interfacing has hastened the development of just such a device: the 8022. By integrating the a-d converter and other useful features, the chip achieves the minimum system cost possible for high-volume controller applications involving analog signals.

The heart of the 8022 is the 8021 general-purpose

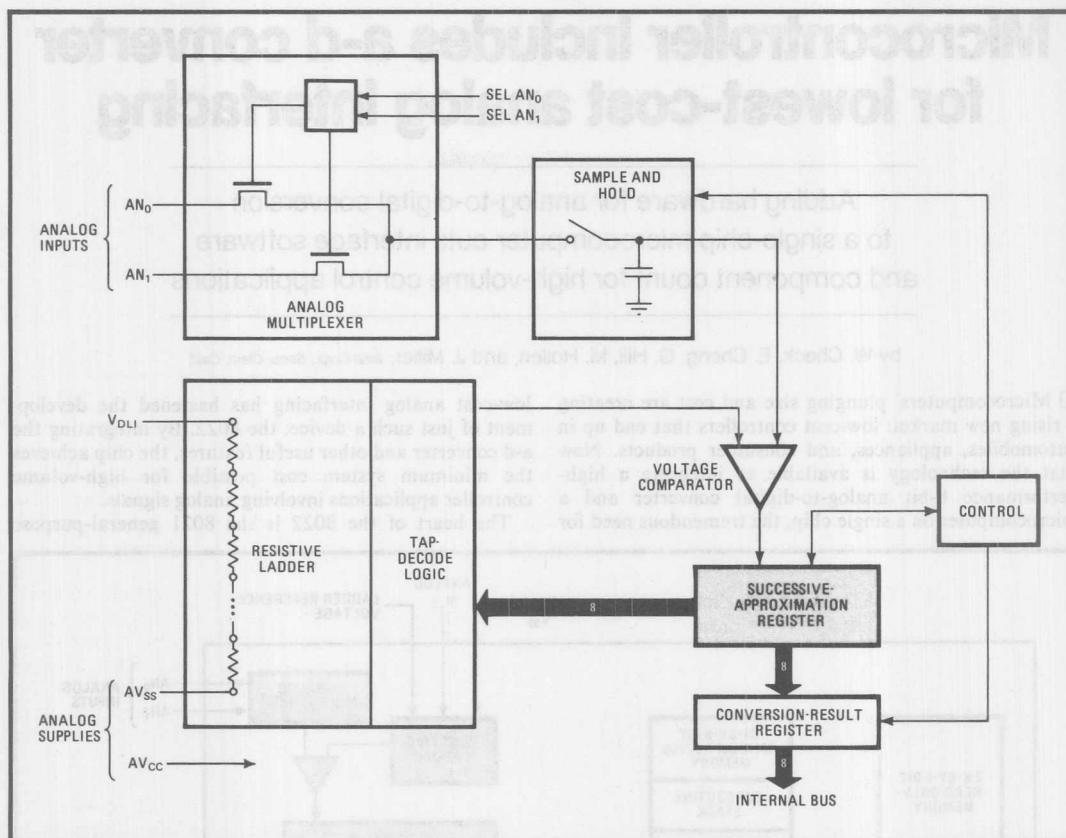


1. All aboard. The first single-chip microcomputer with a built-in 8-bit analog-to-digital converter is Intel's 8022. Around a foundation of the 8021, the chip packs several features that suit it to control applications: two multiplexed analog inputs, a zero-crossing detector, two 7-mA digital outputs that are part of Port 1, and a total of 26 digital input/output lines, eight of which have voltage-comparator inputs.

Reprinted from *Electronics*/ May 25, 1978

Copyright Cahners Publishing Co., Inc. 1977. All rights reserved.

Electronics/ May 25, 1978



2. The converter. The 8022's a-d converter uses successive approximation. A multiplexer selects either of two inputs, which is sampled and held. The successive-approximation register holds a byte that taps off a voltage from a 256-resistor divider through decoding logic. Input is compared with tapped voltage; when the two are equal, the held byte is sent to the conversion-result register.

microcomputer with built-in read-only and random-access memories, which go a long way since many functions are carried out in hardware or require minimal software. The 8021's modular design facilitates its use as a cornerstone for more highly integrated designs like the 8022. This new design, like the 8021, is a member of the MCS-48 family of single-chip microcomputers, and its on-chip a-d converter makes the family even more useful in such high-volume, cost-sensitive application areas as household appliances.

A microcomputer plus

Operating on a single +5-volt power supply, the 8022 contains all the functions necessary for digital processing, plus digital or analog control. On the chip, as diagrammed in Fig. 1, are 2 kilobytes of ROM, 64 bytes of RAM, an 8-bit central processor with more than 70 instructions (a subset of the higher-performance 8048), an internal timer/event counter, a clock and oscillator, the 8-bit a-d converter with two analog inputs, and 26 digital input/output lines.

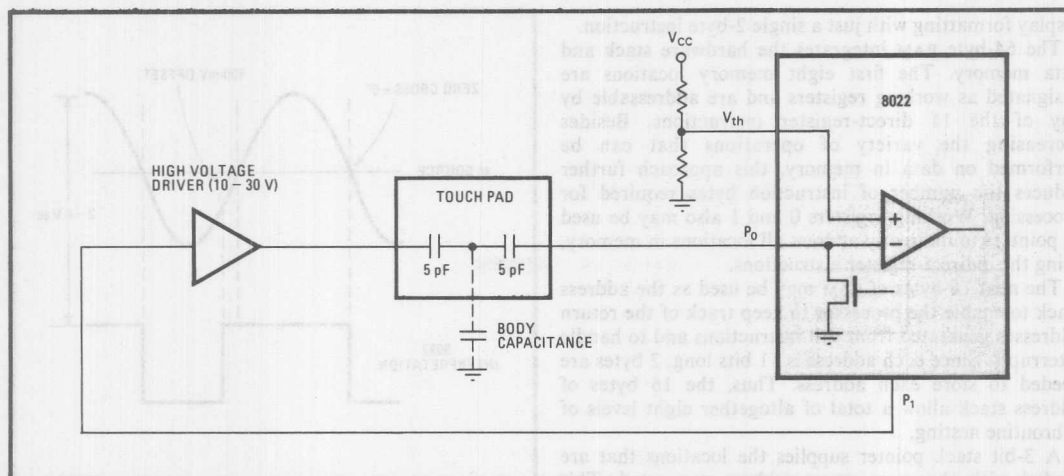
All parts of the a-d converter are integrated onto the chip—no external components are required. Conversion

is performed entirely with hardware by a successive-approximation technique and takes 40 microseconds to complete. The only software is three single-byte instructions: select analog-input 0 (SEL AN0), select analog-input 1 (SEL AN1), and read the analog-to-digital conversion result (RAD).

Flexible I/O lines

The 26 digital input/output lines are organized into three 8-bit general-purpose ports and two test pins, T_0 and T_1 . The three ports are quasi-bidirectional—each line can be programmed for input or output. Adding to the flexibility is an optional mask operation that eliminates the pull-up resistor for the metal-oxide-semiconductor drive transistor on each line, creating an open-drain output. The open drains are useful in driving analog circuits and for certain loads such as keyboards.

Port 0 also has variable-threshold voltage-comparator inputs with a common reference pin (V_{th}). This setup can accommodate such input situations as high noise margins, low-voltage (10-to-15-v) touch switching, and expansion of the analog inputs. Two input/output pins ($P1_0$ and $P1_1$) provide for high-current drive; each sinks



3. Low-voltage touch. Because port 0 has variable-threshold comparator inputs on each of its eight lines, new input configurations are possible, such as this low-voltage touch switch. Touching the panel momentarily pulls down the comparator input. The high-voltage driver, which may be a single transistor or part of a hex driver, then recharges the panel. The port is read by the microcomputer as is any other.

7 milliamperes, more than four times the 1.6-mA load of standard transistor-transistor-logic outputs. In many applications of the 8022, 7 mA can eliminate the need for discrete drive transistors.

The lower half of port 2, in addition to serving as general input/output, may be hooked up as a bus for attaching I/O expander units, such as the 8243, or discrete TTL parts for low-cost I/O expansion. Operations of the 8243 are synchronized by the port-expander strobe pin, a feature that is especially useful for input/output expansions designed with standard transistor-transistor logic gates.

The two test-pin inputs can be tested directly with two conditional-branch instructions. T_0 can interrupt the system, while T_1 also can detect the zero crossing of ac signals—a plus when it comes to firing triacs for phase control of motors.

The a-d converter

The 8022's a-d converter has two multiplexed input channels. Channel selection by either the SEL AN0 or SEL AN1 restarts the conversion sequence. A valid digital value can be read with the RAD instruction during the fourth instruction cycle after a select instruction. Conversions occur continuously, and RAD may be executed at any time with confidence that the sample is no more than 40 μ s old. Typical software for reading two sequential a-d conversions would be:

SEL AN0	Starts conversion
MOV R0,#24	Setup memory pointer
RAD	First conversion to accumulator
MOV @R0,A	Store first value
INC R0	Ready for next conversion
RAD	Second conversion to accumulator
MOV @R0,A	Store second value

As shown in Fig. 2, the conversion hardware itself has

three parts: a series string of resistors, a voltage comparator, and successive-approximation logic. The string of 256 resistors divides the voltage between V_{ss} and V_{DLI} (the reference pin) into 256 voltage steps. This configuration gives the converter inherent monotonicity. Decode logic selects the appropriate tap and transfers that voltage to the comparator block.

The conversion logic

The comparator amplifies the difference between the analog input and the voltage tap. This difference is presented to the successive-approximation logic. Eight comparisons result in a fully converted byte being transferred to the conversion-result register. All comparisons are performed automatically by on-chip hardware; executing the RAD instruction moves the contents of the CRR to the accumulator.

Novel circuit design (see "The a-d converter: how it was done," p. 27) gives the converter 8-bit resolution over the full input range of V_{ss} to V_{cc} . This capability simplifies direct connection to sensors, reduces software, and provides fast, 40-microsecond conversions. The separate power-supply pins complete the analog block and keep the converter isolated from digital-noise sources.

The instruction set

To conserve memory and maximize throughput, most instructions in the 8022 are single-byte and single-cycle; no instructions are longer than 2-byte, two-cycle. The cycle time is 10 μ s.

The overall efficiency of the instruction set is enhanced for control applications by the extensive conditional-branch logic that has been built into the microprocessor. For example, the instruction to decrement a register and jump if not zero (DJNZ) allows loops to be formed in one 2-byte instruction. Similarly, the instruction to move to the accumulator from the current page (MOVP A, @ A) allows table look-up for constants or

display formatting with just a single 2-byte instruction.

The 64-byte RAM integrates the hardware stack and data memory. The first eight memory locations are designated as working registers and are addressable by any of the 11 direct-register instructions. Besides increasing the variety of operations that can be performed on data in memory, this approach further reduces the number of instruction bytes required for processing. Working registers 0 and 1 also may be used as pointers to indirectly address all locations in memory, using the indirect-register instructions.

The next 16 bytes of RAM may be used as the address stack to enable the processor to keep track of the return addresses generated from call instructions and to handle interrupts. Since each address is 11 bits long, 2 bytes are needed to store each address. Thus, the 16 bytes of address stack allow a total of altogether eight levels of subroutine nesting.

A 3-bit stack pointer supplies the locations that are loaded with the next return address generated. This stack pointer is incremented when a return address is stored and decremented when an address is fetched during a return. If an application does not require all eight levels of subroutine nesting, the free portion of the address stack may be used as standard RAM.

Other on-chip features

The 8022 contains its own clock and oscillator circuitry and requires only an external timing control element to generate all internal timing signals. For highly cost-sensitive applications an inductor may be used as this element. If a more precise clock is required, the designer may specify a crystal or external clock for the application.

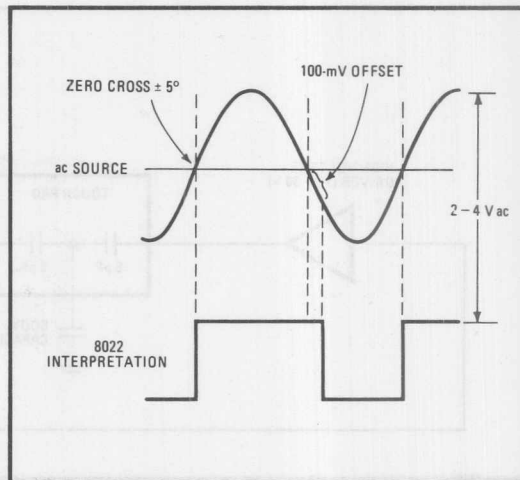
To further reduce the user's system cost and to permit use of the chip in noisy environments, the power-supply tolerance has been increased, permitting a range from 4.5 to 6.5 v. Less filtering and regulation is necessary, therefore, and the microcomputer's immunity to noisy power supplies is greater, as well.

The programmable 8-bit timer/event counter accurately monitors elapsed time, avoiding the software overhead of timing loops. Once it has been loaded with the contents of the accumulator, its divide-by-32 prescaler is incremented for each system clock cycle and at prescaler overflow. A timer flag is set at overflow. Once activated, it can be tested by a conditional-branch instruction to generate an interrupt. Total count capacity is 8,192 instruction cycles or 81.9 milliseconds, for the 10- μ s cycle time.

The timer may also be used as an event counter where the test pin T_1 serves as a counter input. Upon command, the chip will respond to a low-to-high transition on the pin by incrementing its timer.

Comparator inputs

The input/output port 0 of the 8022 has several properties that ease analog interfacing problems. Two of these features are moderate-gain voltage comparators and pull-up resistors on each line that either may serve as standard TTL outputs or may be masked out to give open-drain outputs.



4. Zero-crossing detector. Useful in timing the firing of triacs for ac phase control of appliances or getting a real-time clock, the 8022's T_1 test pin detects the crossing of a waveform's dc level by its rising edge. One hundred millivolts of hysteresis prevents chattering, and the ac frequency is limited to 1 kilohertz.

The comparators are especially handy for troublesome inputs. The comparator at each pin accurately compares that line to the threshold-voltage reference pin, V_{th} , within about 100 millivolts in the range from V_{ss} to $V_{cc}/2$. Allowed to float, V_{th} will bias itself to the digital switch point of the other ports, and port 0 then behaves as a set of conventional digital inputs.

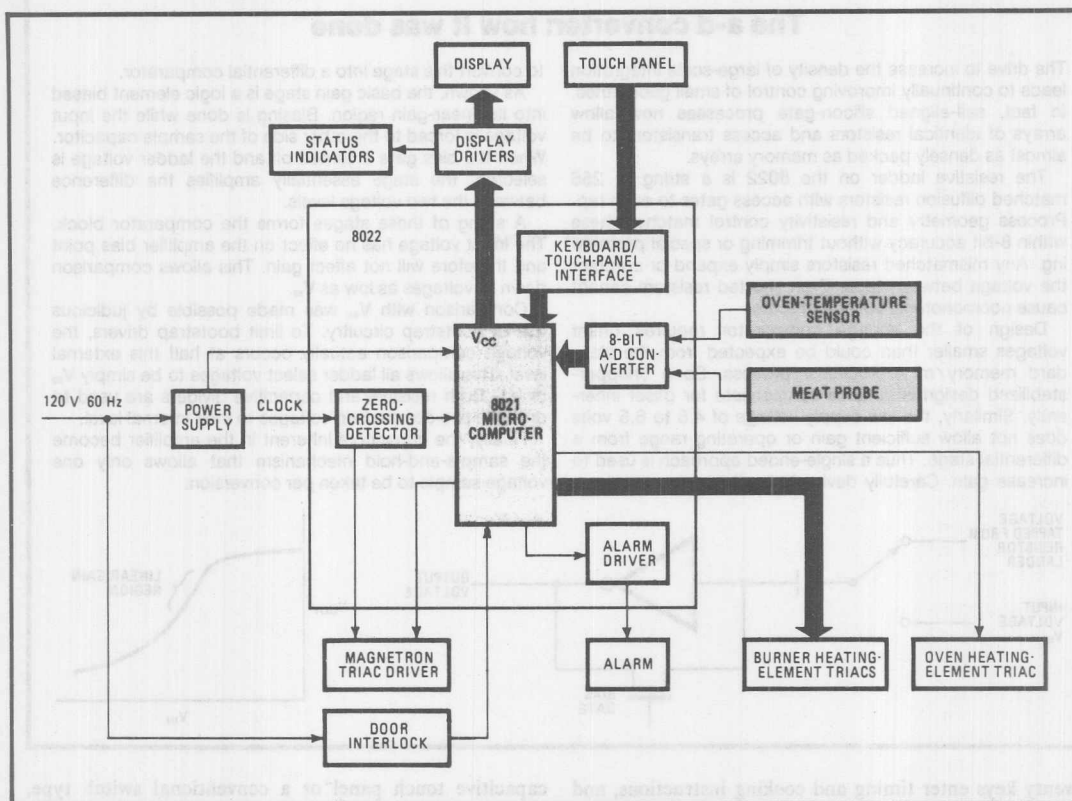
However, the switch point can be both tightly controlled and adjusted by specially biasing V_{th} . Uses for this would include high-noise-margin inputs (up to $V_{cc}/2$), unusual logic-level inputs as from a diode-isolated keyboard, analog-channel extension, and direct interfacing of capacitive touch panels. The comparator action is automatic, and the port is read just as is any other port.

Three advantages

Since the on-chip comparators allow small voltage changes to be detected, a cost-effective and safe touch panel can be built. Many appliances using touch panels have as much as 100 volts at the panel, albeit with extremely low power. The comparators in the 8022, however, permit appliance touch panels to be operated in the 10-to-15-v range.

The advantages of a low-voltage touch panel are three. First, it costs less to generate and switch the lower voltage. Then, since the keyboard operates at below 30 v, it is an Underwriters Laboratories' class II system, which can sharply cut the time required for approval. Finally, the possible product-liability problems associated with high-voltage operation disappear.

Simplified capacitive touch-panel operation is shown in Fig. 3. Contact with the panel drives both the voltage buffer and input to ground. When port 0 is read, a 0 on any line indicates a touched switch. The microcomputer drives the voltage buffer to recharge the panel. Matrix



5. Oven controller. The use of the 8022 is demonstrated in this controller for a combination microwave and conventional oven. The chip needs no assistance in figuring temperatures from thermistors connected to its analog inputs, reading inputs from a touch panel, detecting zero-crossing of ac for firing triacs and gating clocks and timers, direct-driving an alarm, and storing cooking-time instructions.

switch panels may also be sensed by the comparators.

Each pin on port 0 may or may not have an internal pull-up resistor: the option is chosen during selection of the ROM program code. If a resistor is left out for a given pin, the output appears as a true open drain for the range V_{ss} to V_{cc} . There is no temporary low-impedance drive to V_{cc} , as is the case with the remaining quasi-bidirectional ports. With open drains, accurate output waveforms can be generated, and operational amplifiers can be driven directly, for example.

The zero-crossing detector

Although the T_1 test pin on the 8022 may be driven directly by a digital input, it has special circuitry to detect an ac signal crossing its average direct-current level. The signal required for the zero-cross detection mode must be 2 to 4 v peak to peak and have a maximum frequency of 1 kilohertz. It couples to T_1 through an external capacitor.

Figure 4 shows the waveforms for zero-crossing detection. The internal digital state of T_1 is sensed as a 0, until the wave's rising edge crosses the average dc level, when it becomes a 1. The digital transition takes place within a 5° phase from the zero point. The digital level then remains at 1 until the input goes approximately 100 mv

below the zero point on the falling edge. The 100-mv hysteresis keeps noise from causing chattering of the internal signal.

The zero-crossing detection capability allows the applications designer to make the 60-hertz power signal the basis for system timing. All timing routines, including time of day, can be implemented with the signal and just a few conditional jump instructions.

Moreover, since T_1 is also an input to the external event counter, the detection feature may be combined with this counter to interrupt processing at the critical zero-crossing point. Thus the user can trigger phase-sensitive devices, such as triacs and silicon-controlled rectifiers, and use the 8022 in such applications as shaft-angle measurement and speed control of motors—anywhere that the zero crossing of a waveform provides timing information.

An oven controller

The 8022's high level of functional integration provides a single-chip solution to sophisticated, high-volume controller applications that have required relatively expensive multichip designs. An example is a controller (Fig. 5) for a stove with a combined microwave and conventional oven and range-top burners.

The a-d converter: how it was done

The drive to increase the density of large-scale integration leads to continually improving control of small geometries. In fact, self-aligned silicon-gate processes now allow arrays of identical resistors and access transistors to be almost as densely packed as memory arrays.

The resistive ladder on the 8022 is a string of 256 matched diffusion resistors with access gates to each tap. Process geometry and resistivity control matches these within 8-bit accuracy without trimming or special processing. Any mismatched resistors simply expand or contract the voltage between taps. Even shorted resistors cannot cause nonmonotonic voltage outputs.

Design of the voltage comparator requires offset voltages smaller than could be expected from the standard memory/microprocessor process. So a chopper-stabilized design is used to compensate for offset inherently. Similarly, the low supply voltage of 4.5 to 6.5 volts does not allow sufficient gain or operating range from a differential stage. Thus a single-ended approach is used to increase gain. Carefully devised circuit tricks are enough

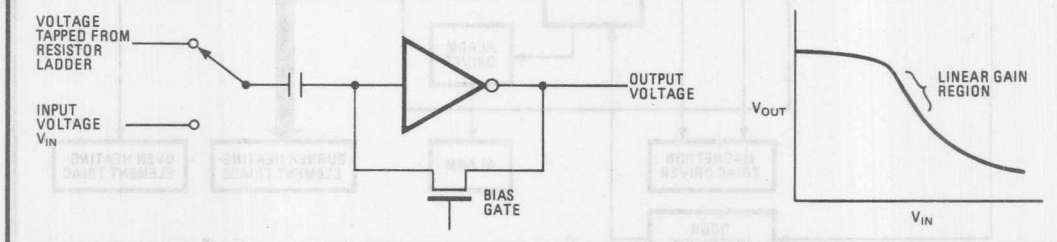
to convert this stage into a differential comparator.

As shown, the basic gain stage is a logic element biased into its linear-gain region. Biasing is done while the input voltage is forced to the other side of the sample capacitor. When the bias gate is turned off and the ladder voltage is selected, the stage essentially amplifies the difference between the two voltage levels.

A string of these stages forms the comparator block. The input voltage has no effect on the amplifier bias point and therefore will not affect gain. This allows comparison down to voltages as low as V_{ss} .

Comparison with V_{cc} was made possible by judicious use of bootstrap circuitry. To limit bootstrap drivers, the voltage comparison actually occurs at half this external level. This allows all ladder select voltages to be simply V_{ss} or V_{cc} . Both resistive and capacitive dividers are used to drop the two comparison voltages to their internal level.

Finally, the capacitors inherent in the amplifier become the sample-and-hold mechanism that allows only one voltage sample to be taken per conversion.



Twenty keys enter timing and cooking instructions, and a four-digit display shows cooking time, temperature, and the time of day. Two temperature-sensing thermistors are employed, one for standard use and the other for microwave use.

While such a system could be controlled by a conventional 4-bit or 8-bit microcomputer, external circuitry would be required to interface the keyboard, convert the analog signals to digital data, drive an audio alarm, and determine the zero-crossing point of the 60-Hz power wave for timing functions and magnetron control. The 8022 reduces this multichip system to a single chip. The computer-plus-converter chip can save the oven maker upwards of several dollars in parts costs.

In this application, the 8022 program memory stores all control programs, cooking and power-cycling algorithms, and timing routines. Its 2-kilobyte ROM is large enough to provide for easy expansion of oven features and product differentiation. The on-chip RAM stores temperatures, power-level and timing settings, and all intermediate computational results.

The analog signals from the conventional temperature sensor and the microwave meat probe feed directly into the two analog inputs on the 8022 without any additional circuitry. What's more, the chip's 8-bit a-d converter gives more accurate temperature sensing than most existing discrete configurations.

The keyboard interfaces directly to the device through port 0. The keyboard in this application can be either a

capacitive touch panel or a conventional switch type, since the 8022 directly interfaces either.

The T_1 pin in the zero-crossing detection mode establishes an accurate time base for all timing routines, including cooking cycles, presetting functions, and time of day. To accomplish this, the chip detects a zero crossing using the two conditional-jump instructions associated with T_1 : JT_1 and JNT_1 . Then it increments a register in data memory, effectively keeping track of elapsed time. Using this technique, a time-of-day routine can be written for most applications in less than 30 bytes of code.

Control of the magnetron

The zero-crossing detection capability also efficiently controls the microwave's magnetron. To minimize current surges through the system, the magnetron should be fired at the peak of the ac wave (90°). To achieve this performance, the 8022 detects the zero crossing point with its T_1 pin and delays the 90° phase shift with the internal timer.

The high-current drive pins, PI_0 and PI_1 , are tied together to directly drive a piezoelectric alarm, which requires 10 to 15 mA of current. The remaining I/O lines are used to drive the display and status indicators, to monitor the door interlock, and to control the triacs that switch the burner and oven heating elements. The internal timer controls the refreshing of the displays and the scanning of the keyboard. □

August, 1978

Microcomputer's On-Chip Functions Ease Users' Programming Chores

William F. Ittner and Jeffrey A. Miller
Microcomputer Components Division

Microcomputer's on-chip functions ease users' programming chores

The one-chip 8022 includes hardware, such as an a-d converter, that combines with the instruction set for easy development of routines

by William F. Ittner and Jeffrey A. Miller, Intel Corp., Santa Clara, Calif.

□ A single-chip microcomputer that incorporates analog-to-digital conversion, comparator inputs, and ac zero-crossing detection is a strong candidate for low-cost, high-volume applications. Moreover, to maintain its front-runner position, the new 8022 has been designed for ease of programming: many common routines are invisible to the user because they are performed in on-chip hardware.

The 8022's instruction set, in conjunction with its hardware features, affords programming ease in the development of routines for translating analog signal levels, monitoring temperatures, reading capacitive-touch-panel inputs, controlling phase-sensitive thyristors, and calculating the time of day. For example, performing an a-d conversion requires software only to select the appropriate analog input; the actual conversion is performed entirely in hardware. This leaves room in the program memory for additional system functions. Furthermore, the instruction set accommodates bit handling, binary and binary-coded-decimal arithmetic, and direct table look-up, and it has extensive facilities for input selection and input-based program jumps.

The 8022 [*Electronics*, May 25, p. 122] is the first general-purpose single-chip microcomputer to offer an on-chip a-d converter. While retaining the 8-bit central processing unit, 64 bytes of random-access memory, clock, zero-crossing detection, and timer/event counter featured in its 8021 predecessor, the new chip doubles the read-only memory to 2 kilobytes and provides comparator inputs on eight input/output lines, five more digital I/O lines (including an extra test pin), full interrupt capability, and two 8-bit a-d input channels. Such a decrease in system component count cannot help but minimize cost and increase reliability.

Easy a-d conversion

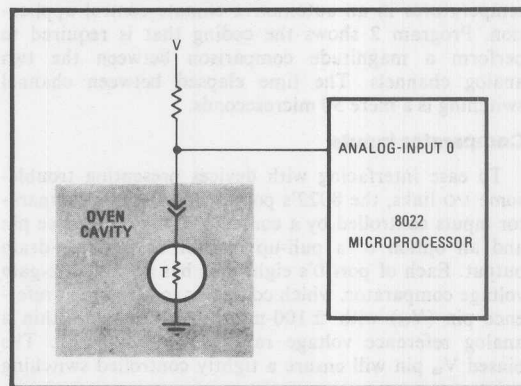
The 8022's a-d converter has two multiplexed channels, selectable with the SEL AN0 (select analog input 0) or SEL AN1 (select analog input 1) instructions. Built-in successive-approximation hardware accomplishes the conversion. The select instructions and the RAD command (read a-d conversion result) are the only software instructions necessary. The select instructions restart the continuously occurring conversion process, but do not affect the conversion-result register. The new valid digital value can be read from the CRR during the

fourth cycle after a select instruction and every fourth instruction cycle thereafter.

An application that points up the advantages of this easy-to-use on-chip converter is monitoring temperature in an oven controller. The temperature is sensed by a thermistor probe located in the oven (Fig. 1). In such a system, noisy analog signals are apt to prevail, obscuring the readings. But since so few instructions are needed for each sampling, a software filtering technique can be added at little expense for increased accuracy. One software filtering method is to average each reading with the previous samples:

SEL AN0	;Start conversion
MOV R0, #30	;Point to storage location of previous a-d sample average
RAD	;Read second sample result
ADD A, @R0	;Add last sample to new sample
RRC A	;Divide by 2
MOV @R0, A	;Store new average

Excessive noise may require averaging of many readings taken over a short period of time. Program 1 illustrates a method of computing the average of 16 readings. In such averaging, it is necessary to select the



1. Talk about simple. To sense temperature with the 8022, all that is needed is a thermistor pulled up to the supply. Simpler yet are the instructions to sense the voltage divider's potential: select analog input 0 (SEL AN0), and read conversion-result register (RAD).

PROGRAM 1: 16 CONSECUTIVE READINGS OF SAME ANALOG INPUT

```

MOV R4, #00    clear temp. MSB result register
MOV R0, #26    set up pointer
MOV @R0, #00   clear result register
SEL AN0        select & start conversion
MOV R2, #16    16 readings

LOOP: RAD       read result
      ADD A, @R0    LSB
      MOV @R0, A    save new LSB
      CLR A
      ADDC A, R4    MSB add carry to R4
      MOV R4, A    save new MSB
      DJNZ R2, LOOP next reading
      SWAP A        MSB into MSN
      XCH A, @R0
      SWAP A
      XCHD A, @R0   divide by 16
                        location 26 in
                        RAM now contains
                        the average value
                        of 16 conversions
                        over a period of
                        1.44 msec

```

PROGRAM 2: MAGNITUDE-COMPARISON ROUTINE

```

SEL AN0        start conversion
MOV R0, #24    set up pointer
RAD            read conversion result
SEL AN1        start other conversion
CPL A
INC A
MOV @R0, A     save first conversion
RAD            read second conversion

ADD A, @R0     add first conversion
                        A equals the differential
                        in ones complement
JZ EQUAL       AN0 = AN1
JC LESTHN      AN0 < AN1
                AN0 > AN1

```

appropriate analog channel only once. Thereafter, a new conversion result is available every four instruction cycles.

Often noise on the analog input is due to 60-hertz ac pickup. To minimize this interference, analog signals should be synchronized with the line voltage, accomplished in the 8022 by the on-board zero-crossing-detection circuitry. The combination of line synchronization with simple filtration renders digital values impervious to line-generated noise.

Signal averaging may be used for more than noise filtering. Since it can be applied to either channel 1 or channel 0 (whichever is selected with a SEL ANx instruction), each channel may monitor different functions in one system, such as temperature and pressure in a process-control application. The fast a-d conversion time permits rapid switching between the two channels.

A similar software technique permits measuring the same variable in two different locations and comparing the two results, as in checking the internal and external temperatures in an automotive climate-control application. Program 2 shows the coding that is required to perform a magnitude comparison between the two analog channels. The time elapsed between channel switching is a mere 50 microseconds.

Comparator inputs

To ease interfacing with devices presenting troublesome I/O links, the 8022's port 0 incorporates comparator inputs controlled by a common voltage-reference pin and an option of a pull-up resistor or an open-drain output. Each of port 0's eight pins has a moderate-gain voltage comparator, which compares to a common reference pin (V_{th}) with ± 100 -millivolt accuracy, within a analog reference voltage range of V_{ss} to $V_{cc}/2$. The biased V_{th} pin will ensure a tightly controlled switching point.

A typical use for port 0 is in the interfacing with the capacitive touch panels on microwave ovens and other new appliances. A touch-panel switch consists of two capacitors in series. One lead is attached to a high-

voltage buffer (10 to 30 volts). The other is attached to the port 0 sense input. As a finger touches the common point, the drive signal is shunted by body capacitance, attenuating the signal reaching the input.

Low-voltage touch-panel operation (less than 30 v) is possible, since the comparators allow small voltage changes to be detected. Most of the present touch-panel designs require a 50-to-100-v drive on the touch panel.

Capacitive touch panels can be multiplexed in the same manner as can mechanical keyboards (Fig. 2). The vacuum fluorescent display and the touch panel are integrated to optimize hardware through shared high-voltage buffers.

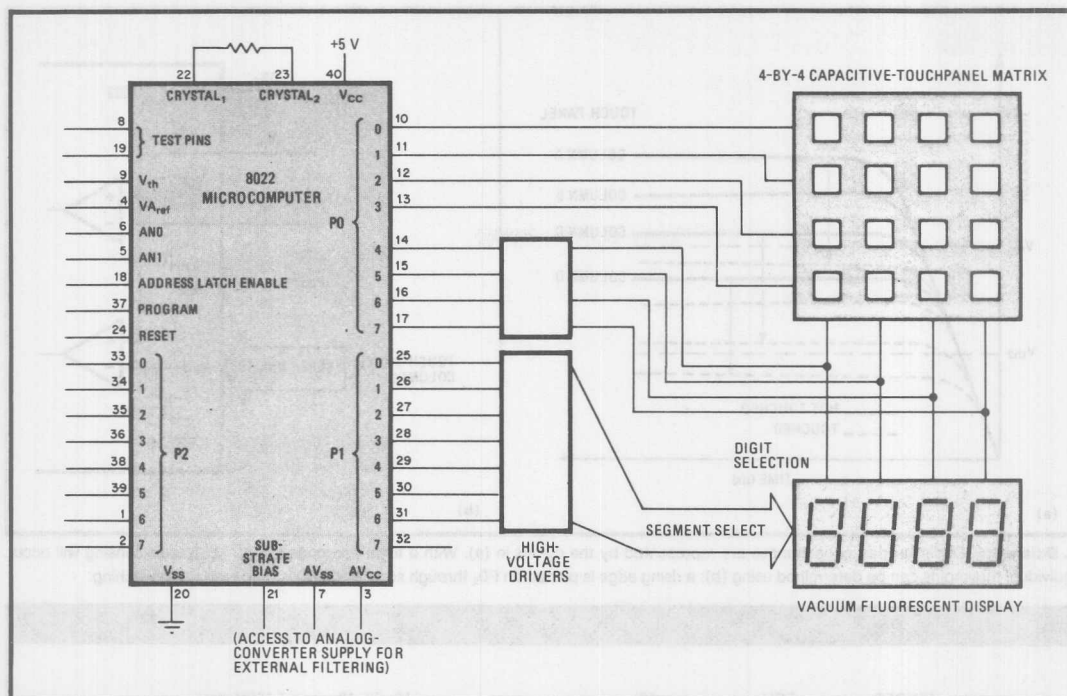
Program 3 lists the software that is necessary to refresh the display and scan the touch-panel matrix. This routine could be adapted to serve as part of a timer/interrupt scheme that would generate an interrupt at precise intervals for a flicker-free display. Another portion of the software would check for any touched input pads, test for valid entry, and enter key-depression codes into the main program.

Correcting pad imbalance

A common problem with capacitive touch panels is their imbalance. Layout process, aging, and surface impurities all cause the capacitance to vary from touch pad to touch pad, resulting in a family of curves (Fig. 3a) of voltage levels from each column of touch pads reaching the sense inputs. As the curves show, if threshold voltage V_{th1} alone were used, one column would always appear touched; if V_{th2} were used exclusively, three of the columns would never appear touched.

To compensate for such varying capacitance levels, the on-chip analog-input circuit may be used to allow multiple input voltage levels. Figure 3b depicts the 8022 version of such a circuit. $AN0$ and V_{th} are tied together with a capacitor to line 0 of port 0. The pull-up resistor option is used on line 0 to provide an RC timing network connected to $AN0$, V_{th} , and $P0_0$. The remaining seven lines of port 0 are the sense lines for the touch panel and use the open-drain-output option.

Handling the different voltage levels would begin with initializing the data by plotting the family of curves with the $AN0$ input. This is done by writing a 0 to $P0_0$ (grounding $P0_0$) which initializes V_{th} to 0 v. A logic 1 is then written to $P0_0$, which begins to pull the RC network



2. Multiplexed touch panel. A capacitive touch panel and high-voltage display may be combined in much the same way as a mechanical keyboard and light-emitting-diode array. To save hardware, an obvious choice is to share the high-voltage drivers.

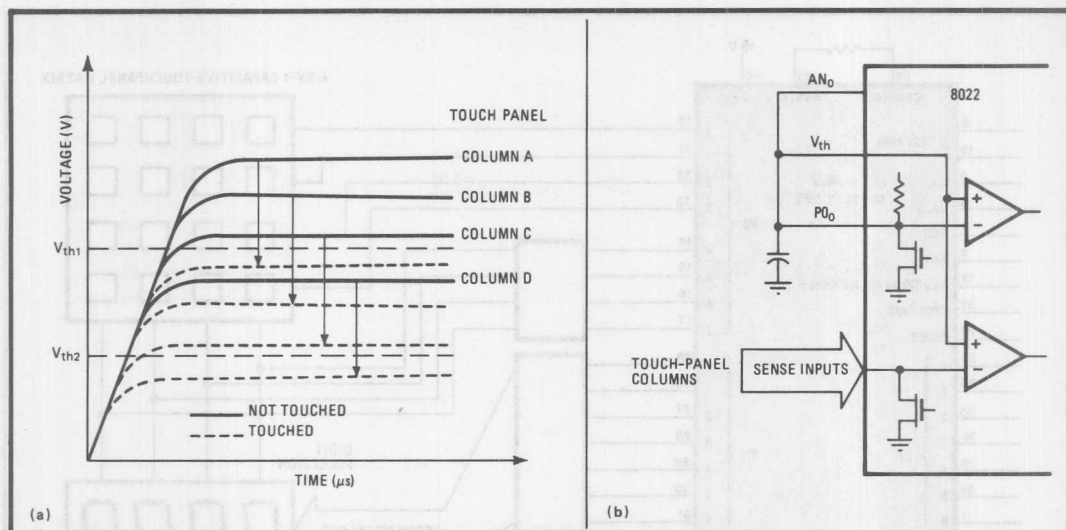
PROGRAM 3: DISPLAY REFRESH KEYBOARD SCAN ROUTINE			
	CLR OUTL OUTL	A P1, A P0, A	turn off segment drivers turn off digit drivers and panel strobes and initialize sense input to ground
	MOV OUTL ORL OUTL IN	A, #0FH P0, A A, R3 P0, A A, P0	float sense inputs new strobe position turn on strobe read sense inputs
	MOV	R4, A	save sense inputs
	MOV MOV OUTL	R0, #D1SP4-1 A, R3 P0, A	RAM location of MSB of 7 segment pattern strobe position into A GND sense inputs
LOOP:	RLC INC JNC MOV OUTL	A R0 LOOP A, @R0 P1, A	rotate digit strobe into carry next digit location loop until carry retrieve pattern from RAM new segment pattern

toward 5 v. As V_{th} ramps upward, the sense lines are monitored for input changes, which will go from 1 to 0 as the not-touched voltage curve for each input is intersected. As the changes occur, the a-d value for each sense line can be read and stored.

Thus the threshold reference voltage for each sense line can be determined by establishing the not-touched voltage levels and by placing the threshold reference voltage below this level. As each row of the keyboard is

scanned, the RC network is initialized to 0 v and ramps upward, varying the V_{th} level. The a-d converter monitors this level looking for the calculated threshold points. As the points are intersected, the corresponding sense inputs are read, with a 0 indicating a touched input and a 1 indicating a not-touched input. Thus, a multiplexed capacitive touch panel can be scanned and balanced without adding external components to the inputs.

For systems requiring more than two analog inputs to



3. Balancing. Dissimilar pad capacitances are represented by the curves in (a). With a fixed reference (V_{th1} or V_{th2}), false sensing will occur. Individual thresholds can be determined using (b): a rising edge is placed on $P0_0$ through software; AN_0 is then read until switching.

PROGRAM 4: 90° PHASE-ANGLE ROUTINE			
NINDEG	EQU	13	13x32x10 usec = 4.160M msec
LOC 7:	XCH	A, R7	save Acc and get flag byte
	INC	A	
	JNZ	NINETY	is it zero cross or 90 deg.
	MOV	A, #NINDEG	zero cross"
	MOV	T, A	set up for ninety degrees interrupt
	STRT	T	
	XCH	A, R7	load R7 with non-FF number
	RETI		and restore A"
NINETY:	IN	A, P1	
	MOV	A, #11101111B	set P14 low (TRIAC PORT)
	OUTL	P1, A	
	MOV	A, #0FFH	set up for zero cross interrupt
	MOV	T, A	next time"
	STRT	CNT	
	XCH	A, R7	load FF into R7 and restore Acc
	RETI		

the 8022, port 0's comparator may be reconfigured to permit forming of pseudo-analog inputs from variable-threshold digital inputs. The hardware configuration can be identical to that of Fig. 3b. In this scheme, sense inputs act as additional analog inputs of less accuracy than AN_0 and AN_1 (about 6 bits).

The sequence for implementing the extension is essentially the same found in the variable-threshold touch panel. As V_{th} ramps upward, a port 0 bit is monitored for a change from 1 to 0. At the change, AN_0 is read,

corresponding to the value of the analog input into port 0 (with some error due to the time lag, which can be subtracted). This configuration can be utilized when it is possible to trade off accuracy for cost improvements: one analog input with 8-bit accuracy and seven analog inputs with 6-bit accuracy. Such may be the case in a range controller that monitors temperature in two ovens, a meat probe, and two of the four burners, all to an accuracy within 10°F.

To establish a reliable time base and to switch ac

	ORG	7	T1 timer interrupt vector
	MOV	A, #0FFH	initialize timer
	MOV	T, A	
	MOV	R0, #TIME0 - 1	TIME0 = LSB of timer register
	MOV	R1, #TABLE	LSB of ROM look-up
LOOP:	INC	R0	point to next byte
	MOV	A, @R0	retrieve BCD byte
	ADD	A, #1	increment
	DA	A	decimal adjust
	MOV	@R0, A	restore BCD byte
	MOV	A, R1	test for carry
	MOVP	A, @A	using table entry
	XRL	A, @R0	and"
	INC	R1	bump pointer
	JNZ	DONE	no carry, wait for next tick
	XCH	A, @R0	carry, set digit pair to 00
	ANL	A, #1	was overflow hours?
	JZ	LOOP	no, increment next byte
	MOV	@R0, A	yes, set hours to 1
DONE:	RETI	1	
TABLE:	DB	60 H	sixtieth
	DB	60 H	minutes
	DB	13 H	hours use 25 for 24 hour operation

loads, the 8022 has circuitry built into the T₁ pin to detect an ac signal crossing its average dc level. The switching is at predetermined points of the sine wave to reduce inrush currents or radio-frequency interference.

Zero-crossing detection

There are several methods by which software can monitor the input. The simplest method involves the jump instructions JT₁ and JNT₁, which correspond to jump on T₁ high, and jump on T₁ low, respectively. The rising edge of the T₁ input is the most accurate: the falling edge contains 100 mv of hysteresis to increase noise margin. The two jump instructions can be used back to back to find this zero-crossing point:

```
HERE1: JT1 HERE1 ;Wait here if line high
HERE2: JNT1 HERE2 ;Wait here if line low
; Zero cross
```

To reduce loop time, the T₁ pin may also be coupled to the event counter. The start-counting instruction couples the rising edge into the 8022's internal 8-bit timer. With each rising edge, the timer increments by one, and when it increments from FF Hex to 00, an overflow flag is set. If the interrupt line is activated, an interrupt vector at location 7 will occur. Since the timer may be preloaded with any value, it is possible to cause an interrupt to occur on the next zero crossing rather than waiting in the jump loop.

The following routine will initialize the timer to accomplish this. All other processing may be performed

while waiting for the zero crossing. The timer could be reloaded with FF Hex during the interrupt routine to generate an interrupt on each zero crossing.

```
MOV A, #0FFH ;Full count into accumulator
MOV T, A ;Load timer
STRT CNT ;T1 pin is source to timer
EN TCNTI ;Enable timer interrupt
```

Of course, the zero crossing is not always the best point to gate a control device. For example, an application involving a highly inductive device such as a magnetron transformer will produce inrush currents that are at their maximum at the zero-crossing point.

Low inrush

To minimize inrush in such a system, the triac is turned on at a 90° phase angle in the 60-Hz sine wave. Program 4 provides the software necessary to accomplish this task. The timer detects the zero-crossing point and times out to the 90° point, where the leading current will just be at a minimum. An interrupt occurs at both the zero and 90° points to prevent interference with normal processing. Both interrupts use the same interrupt vector location. Software determines the source of the interrupt and acts accordingly.

Another use of the T₁ input is generating the timing base for a time-of-day routine. The software implementing this routine is in program 5. The time parameters listed in the accompanying data table could be modified to accommodate either 12- or 24-hour operation. □

8-3	Introduction
8-3	Product Overview
8-5	Product Features
8-5	System Clock
8-5	Interrupt Mode
8-5	Crystal Mode
8-5	Watch Dog
8-7	Timer/Counter
8-8	Test and Interrupt Inputs
8-9	Zero Cross Detect
8-10	Analog to Digital Converter
8-12	Software Noise Reduction
8-13	Port 0 Comparator Inputs
8-15	Application Ideas
8-17	Power Supply Controller
8-17	DC Motor Control
8-18	Automotive Dashboard
8-19	Desktop Timer
8-19	Conclusion

MARCH 1979

Designing With Intel's 8022 Microcomputer

Will Ittner
MCO Applications

Designing With Intel®'s 8022 Microcomputer

Contents

Introduction	8-3
Product Overview	8-3
Product Features	8-5
System Clock	8-5
Inductor Mode	8-5
Crystal Mode	8-5
Which One?	8-5
Timer / Counter	8-7
Test and Interrupt Inputs	8-8
Zero Cross Detect	8-9
Analog to Digital Converter	8-10
Software Noise Rejection	8-12
Port 0 Comparator Inputs	8-13
Application Ideas	8-15
Power Supply Controller	8-15
DC Motor Control	8-17
Automotive Dashboard	8-18
Darkroom Timer	8-19
Conclusion	8-19

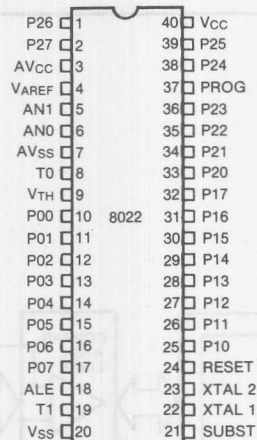


Figure 1. Pin Configuration

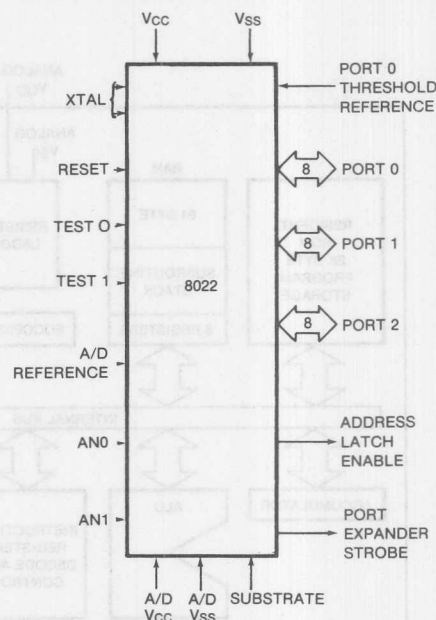


Figure 2. Logic Symbol

INTRODUCTION

Taking advantage of the latest advances in silicon technology, Intel has developed a complete control system on a chip, the 8022, the first 8-bit microcomputer with an A/D converter on-chip. Whereas in the past microcomputers relied on external circuits for analog interfacing, it is now possible to build a one chip control system with analog interfacing, digital interfacing, and computer processing capabilities. Tackling the high volume, low cost controller market, the Intel 8022 microcomputer fits cost and space sensitive applications such as automobiles, appliances, and consumer products previously dominated by electromechanical controls. Its use, however, is not confined only to these applications. In medium volume applications, the 8022 provides the system designer with a simplified solution to many control problems. No longer is it necessary to expend valuable engineering time designing wheel spokes and axles; the whole cart is available.

This note is intended to answer some design questions concerning the 8022 and to suggest to the reader possible applications and system configurations. The reader should refer to the 8022 Data Sheet for electrical specifications and details. It is also suggested that the reader consult with the MCS-48 User's Manual (July 1978 or later) for a complete description of the entire MCS-48 family of microprocessors of which the 8022 is the newest member.

The note is divided into two main sections. The first is a product description of the 8022, including a detailed discussion of the main features, their characteristics and how to use them. The second section discusses several possible applications, their configurations and design considerations.

Product Overview

The heart of the 8022 is the Intel 8021, a general purpose single chip microcomputer, which is a lower performance, lower cost version of the 8048. Added to this central core are interrupts, additional I/O, and linear functions. Like the 8021, the 8022 is designed to operate over a power supply range of 4.5 to 6.5 volts.

The 8022 instruction set contains over 70 instructions and is a subset of the 8048 instruction set. To conserve memory and maximize throughput, most instructions are single-byte, single-cycle. No instructions are longer than two-byte, two-cycle. The instruction cycle time is 10 microseconds at a 3MHz clock rate. Extensive conditional branch logic is built into the processor to increase the overall efficiency of the instruction set for control applications. As examples, the DJNZ instruction (decrement register and jump if not zero) allows loops to be formed in just one instruction and the MOVP A, @A allows single instruction table look-up of constants from program storage. Program storage in the 8022 consists of 2048 eight bit bytes of mask programmable ROM.

Hardware stack and data memory are integrated in the 64 byte RAM to enhance processing flexibility and memory utilization. The first eight RAM locations are designated as working registers and are directly addressable by any of the 11 direct register instructions. Besides increasing the variety of operations that can be performed on data in memory, this approach further reduces the number of instruction bytes required for processing. In addition to being used as working registers, Registers 0 and 1 can be used as Pointer registers to indirectly address all locations in memory using the indirect register instructions.

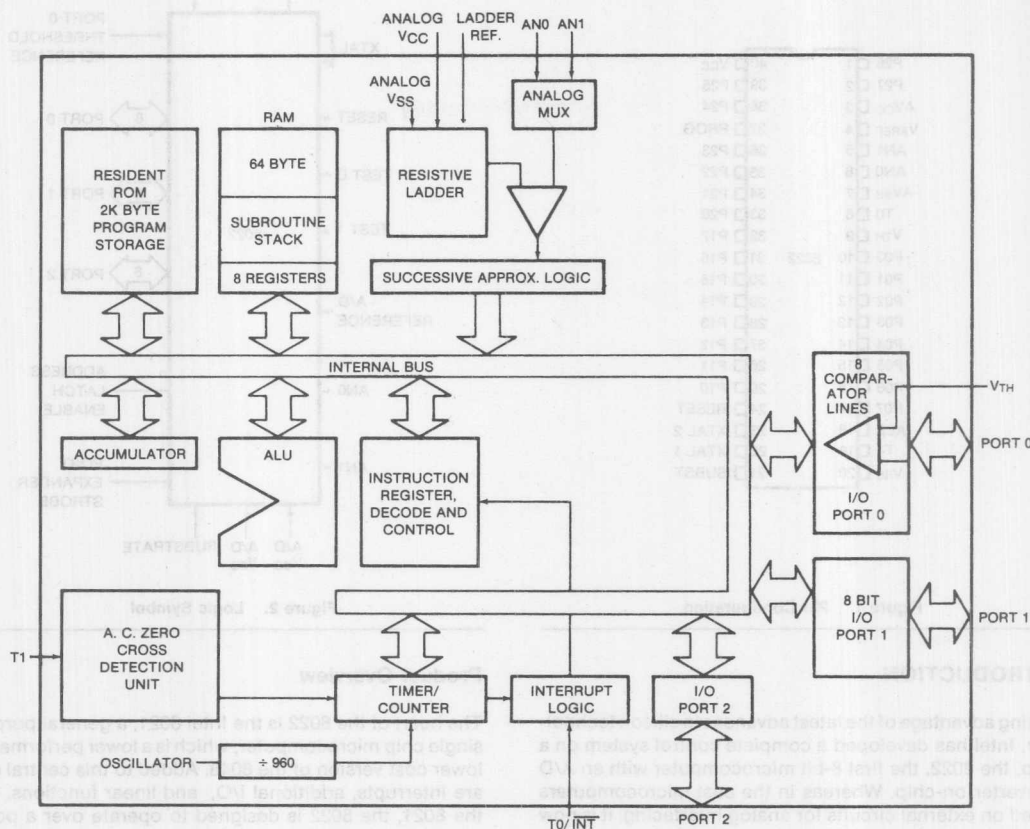


Figure 3. 8022 Block Diagram

The next 16 bytes of RAM may be used as the address stack to enable the processor to keep track of the return addresses generated from instructions and in handling interrupts. Since two bytes are needed to store each address, the 16 bytes of address stack allow up to a total of eight levels of subroutine nesting. A 3-bit stack pointer supplies the address of the locations to be loaded with the next return address generated. This stack pointer is incremented when a return address is stored and decremented when an address is fetched during a subroutine or interrupt return. If all eight levels of subroutine nesting are not required by an application, the unused portion of the address stack may be used as standard RAM.

The 8022 has an extremely flexible and powerful I/O structure. The 26 digital I/O lines are configured into three 8-bit general-purpose ports and two test pins, T0 and T1. All three ports are quasi-bidirectional, meaning all lines are useable as inputs or outputs on a line-by-line basis under software control.

To increase the user's flexibility, any line of Port 0 can also be designated an open drain output by removing the pullup device present on the line via mask option. This is useful in driving analog circuits and interfacing to high impedance digital I/O. In addition to the open drain option, Port 0 has voltage comparator inputs with a common reference pin (V_{TH}). In appliance control and other applications, this allows direct glass touchpanel interfacing with relatively low voltage (10-15V) drive, thus limiting product liability problems and easing U.L. approval. The Port 0 comparator inputs are also generally useful in many other ways from expanding analog inputs to maximizing margin on noisy signals.

To further increase user flexibility and reduce system cost, two I/O pins (P10 and P11) have been designated as high current drive pins with the ability to sink 7ma each, instead of the standard TTL load of 1.6ma. This can eliminate the need for discrete drive transistors in many applications.

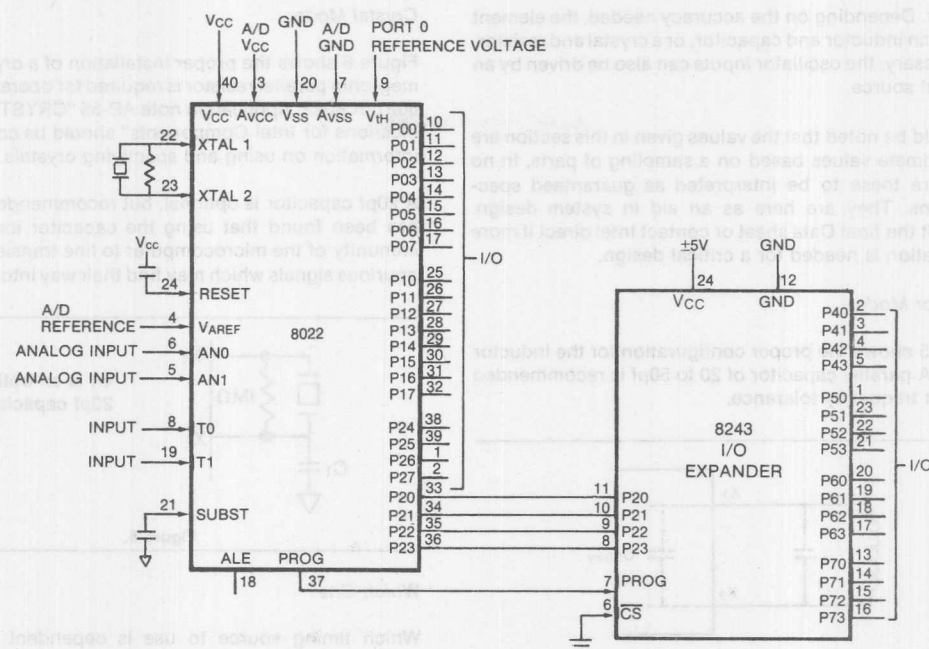


Figure 4. Adding an I/O Expander to the 8022

The lower half of Port 2, in addition to serving as a general-purpose I/O port, is used as a "bus" for attaching the Intel 8243 I/O expander units. The Port Expander Strobe is used in conjunction with Port 2 to synchronize the 8243 operations. Figure 4 shows such a configuration.

Note that the quasi-bidirectional structure and the Port 2 expansion bus are consistent with all MCS-48 products and are fully described in the MCS-48 User's Manual.

Frequently in control applications, the state of one or two signals must be monitored so that a fast response can be accomplished. The 8022's two test pins offer this capability. Both test pins, T0 and T1, are directly testable via two conditional branch instructions. The T0 pin can also cause an interrupt. The T1 pin, in addition to being directly testable, has the ability to detect the zero crossing of slowly moving AC inputs. This is useful in controlling 50/60Hz power. It also enables the 8022 to precisely control phase sensitive devices, such as triacs and SCRs. Again external circuitry is reduced.

The 8022 contains its own clock and oscillator circuitry and requires only an external timing control element to generate all internal timing signals. An inductor, a crystal, or an external clock may be used as the timing control device.

The programmable 8-bit timer/event counter enables the user to accurately monitor elapsed time by providing a hardware replacement for software overhead such as timing loops. Total count capacity is 8192 instruction cycles or 81.9 msec at a 10 microsecond cycle time. The timer may also be used as an event counter where the Test

1 input serves as a counter input. After a STRT CNT command, low to high transitions on the T1 pin will cause the timer/counter to be incremented. When the timer counter overflows (FFH to 00), the timer flag will be set and an interrupt generated if enabled.

The analog to digital converter is designed to simplify and cost reduce interfacing to analog sources. All parts of the converter are integrated onto the chip, with the exception of the voltage reference. Conversion is completely hardware controlled using a successive approximation technique and occurs in four instruction cycles or 40 microseconds. Three single byte instructions, SEL AN0 (select analog input 0), SEL AN1 (select analog input 1), and RAD (read A/D conversion result) are added to the 8021 instruction set to allow the programmer to interface to the converter conveniently.

Product Features

This next section will delve deeper into some of the functions which comprise the 8022 architecture. Chip architecture will be discussed along with design considerations, software routines, and hardware configurations. The specific items covered are CPU timing, the Timer/Counter, the TEST and Interrupt inputs, Zero Cross detection, the A/D converter, and the Port 0 comparator inputs.

System Clock

One of the first considerations in the system design is what frequency source should be used. The on-board oscillator can use a variety of elements to determine system fre-

quency. Depending on the accuracy needed, the element can be an inductor and capacitor, or a crystal and resistor. If necessary, the oscillator inputs can also be driven by an external source.

It should be noted that the values given in this section are approximate values based on a sampling of parts. In no case are these to be interpreted as guaranteed specifications. They are here as an aid in system design. Consult the final Data sheet or contact Intel direct if more information is needed for a critical design.

Inductor Mode

Figure 5 shows the proper configuration for the inductor mode. A parallel capacitor of 20 to 50pf is recommended for best frequency tolerance.

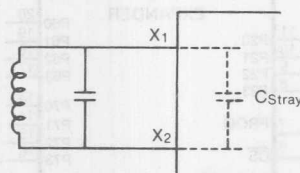


Figure 5.

Table 1 shows the effects of changes in parameters based on a sampling of parts. Part to part input capacitance differences (Cstray) will effect the tolerance. A less than 0.2% part to part tolerance can be expected with a parallel capacitance of 50pf. (see fig. #5). An additional 0.5% variation comes about when only 20pf is used in the tank circuit. This is because the stray capacitance in the 8022 and the PCB becomes a larger proportion of the total capacitance.

Vcc =	4.5v	5.5v	6.5v
f =	±0.2%	0	≈0.2%
Temp =	-40°C	25°C	85°C
f =	±0.6%	0	≈0.6%

Table 1. Inductor Mode

To determine the inductance and capacitance required for a given frequency, the equation

$$f = \frac{1}{2\pi\sqrt{LC}}$$

can be used. Due to the effects of stray capacitance the calculated frequency may be slightly high. It should be noted that the tolerances given in Table 1 do not include the tolerances of the inductor and capacitor used in the system. Mathematical analysis of the above equation will show that the frequency will change roughly proportional to the tolerances of L and C on a worst case situation. That is if both L and C are ±5% parts, the frequency will vary approximately ±5%.

Crystal Mode

Figure 6 shows the proper installation of a crystal. A one meg-ohm parallel resistor is required for operation with an 8021 or 8022. Application note AP-35 "CRYSTALS: Specifications for Intel Components" should be consulted for information on using and specifying crystals.

A 20pf capacitor is optional, but recommended, on X2. It has been found that using the capacitor increases the immunity of the microcomputer to line transient noise or spurious signals which may find their way into the system.

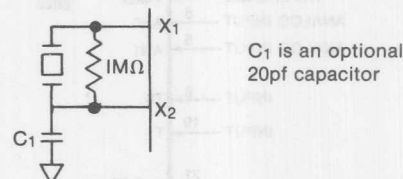


Figure 6.

Which One?

Which timing source to use is dependent on several factors. In most applications cost is of primary importance. The lowest cost device, but one which still gets the job accomplished, is the logical choice. Selecting the device which gets the job accomplished is the next task.

A Case Study

To exemplify the design tradeoffs in choosing a timing element consider the detection of 50Hz or 60Hz line frequency as may be needed in many consumer products being sold in the U.S. and overseas. Traditionally two products are produced, one for the U.S. market and one for the overseas market. A jumper selection to tell the processor which frequency source is being used is the only difference. This costs one I/O pin plus the costs of insertion and inventorying two products. All of these costs can be saved by allowing the processor to compute which frequency is coming in on the T1 pin. Figure 7 lists the software which could be used during a power-up routine to determine whether 50Hz or 60Hz timing should be used.

The timer is used to time the interval of one line cycle. If everything were perfectly accurate, one count would equal 50Hz while another count would equal 60 Hz, but it's not. The power company frequency may shift slightly, plus the 8022 oscillator may drift as discussed earlier. The maximum allowable oscillator change must be calculated from the input source. Assuming the power companies may drift ±2 cycles, then the processor must be able to detect a difference of 58Hz-52Hz = 6Hz or less than 10.3% change. This means that the oscillator frequency itself cannot change more than 10.3% or ±5.15%. The crystal would definitely work but may be overkill. The Inductor/capacitor combination could be the most economical solution.

The equation

$$\frac{1}{\frac{LF}{1 \times 30 \times 32}} = \text{count}$$

where LF = line freq,
f = osc. freq.

will give the value of the time at the end of one line cycle. Plugging in the values for an oscillator of 3MHz $\pm 5\%$ and a ± 2 cycle deviation in line frequency, the counter will yield counts of:

$$47-56 = 60\text{Hz}$$

$$57-68 = 50\text{Hz}$$

Inductor and capacitor components could be picked to yield the required tolerance, saving the costs previously mentioned.

Timer/Counter

An 8-bit interval timer/counter is available to enable the user to keep track of time elapsed or number of events occurred while normal program execution and flow continues. The Auto 50/60Hz detection routine previously discussed is one of many possible applications of the timer/counter.

The timer/counter consists of a divide by 32 prescaler (only used in the timer mode) and an eight bit main timer. The STRT T command clears the prescaler and thereafter it increments once each 30 input clocks (once each single cycle instruction, twice each double cycle instruction). At the (11111) to (00000) transition the timer is incremented. A timer overflow from (FFH) to (00H) will set the timer flag along with the timer interrupt, if enabled (see below). A conditional branch instruction (JTF) is available for testing

LOC	OBJ	LINE	SOURCE STATEMENT
		1	;
		2	;
		3	;
		4	;
		5	PWRUP:
0000	27	6	CLR A ;CLEAR ACCM
0001	0414	7	JMP PWRDET ;JUMP AROUND INTERRUPT ROUTINES
		8	;
		9	;
		10	;
		11	;
		12	;
0014		13	ORG 20
		14	;
		15	MEASURE ONE LINE CYCLE
		16	;
		17	;
		18	PWRDET:
0014	5614	19	JT1 PWRDET ;WAIT FOR NEXT RISING EDGE
		20	LINLOW:
0016	4616	21	JNT1 LINLOW ;WAIT FOR NEXT RISING EDGE
0018	62	22	MOV T,A ;CLEAR TIMER
0019	55	23	STRT T ;START TIMER
		24	LINEHI:
001A	561A	25	JT1 LINEHI ;WAIT HERE FOR LINE TO GO LOW
		26	RISEDOG:
001C	461C	27	JNT1 RISEDOG ;WAIT HERE FOR RISING EDGE
001E	65	28	STOP TCNT ;STOP TIMER AT END OF ONE LINE CYCLE
001F	42	29	MOV A,T ;READ TIMER VALUE
0020	03D1	30	ADD A,#-47 ;SUBTRACT 47
0022	E62C	31	JNC ORANGE ;ERROR-NOT WITHIN RANGE
0024	03F6	32	ADD A,#-10 ;SUBTRACT 10
0026	E62C	33	JNC HZ60 ;JUMP TO 60HZ ROUTINE
0028	03F4	34	ADD A,#-12 ;SUBTRACT 12
002A	F62C	35	JNC ORANGE ;ERROR-NOT WITHIN RANGE
		36	HZ50:
		37	;
		38	HZ60:
		39	;
		40	ORANGE:
		41	;
		42	;
		43	;
		44	;
		45	;

Figure 7.

this flag, the flag being reset each test. This instruction must also be used to initialize the timer overflow flag after a RESET, as RESET does not perform this function. Total count capacity for the timer is $2^5 \times 2^8 = 8192$ or 81.9 ms at a 10 micro second cycle time. Contents of the timer are moved to the accumulator by the MOV A,T instruction without disturbing the counting process. Conversely, the MOV T,A instruction loads the timer with the contents of the accumulator. Notice that the 8-bit timer can be read from and written to. The prescaler, however, can not. It is a separate 5-bit counter which is cleared only by a STRT T command.

The timer may also be used as an event counter. After a STRT CNT command the 8022 will respond to low-to-high transitions on the Test 1 pin by incrementing the timer. Transitions can occur no faster than once each three instruction cycles (every 30 microseconds when using a 3 MHz clock)—there is no minimum frequency. In this mode the prescaler is not used. The timer will contain the number of positive transitions occurring on T1 since a STRT CNT command.

The timer and event counter functions are mutually exclusive. Counting or timing may be started (STRT CNT, STRT T) or stopped (STOP TCNT) under program control.

The T1 pin, besides being an input to the counter, can also function as a testable input, detect the zero crossing of an AC signal, and interrupt processing. These functions, as well as those of the Test 0 pin and the interrupt structure, will be discussed in the next section.

Test And Interrupt Inputs

In addition to the 24 general purpose I/O lines which comprise ports 0, 1, and 2, the 8022 has two special inputs, T0 and T1, which are testable via conditional jump instructions. These pins allow inputs to cause program branches without the necessity to load an input port into the accumulator. The instructions JT0, JNT0, JT1, JNT1 will cause program flow to be modified depending on the state of the T0 or T1 pin. For instance, JT0 will cause a jump to the specified address if the T0 pin is high (a 1 level). Conversely, JNT0 will jump if T0 is low (a 0 level). If the jump does not occur, program flow continues with the next instruction.

The Test 0 pin serves as an external interrupt input as well as a testable input. An interrupt sequence is initiated by applying a low "0" level input to the T0 pin when the external interrupt is enabled (EN I). The interrupt is level triggered and active low to allow "WIRE ORING" of several interrupt sources at the input pin. When an interrupt is detected it causes a "call to subroutine" to location 3 in program memory as soon as all other cycles of the current instruction are complete. At this time, the program counter contents are saved in the program counter stack, but the remaining status of the processor is not.

Unlike the 8048, the 8022 does not contain a program status word. Thus, when appropriate, the carry and auxil-

ary carry flags must be saved by the software, as must be the accumulator. The routine shown below saves the accumulator and the carry flags.

Instructions	Comments
MOV R6,A	;save accumulator in register 6
CLR A	;clear accumulator
DA A	;convert carry flags into sixes
MOV R7,A	;save representation of carry flags

The end of an interrupt service subroutine is marked by the execution of a Return from Interrupt instruction (RETI). Prior to returning from the interrupt subroutine however, the status of the accumulator and the carry flags must be restored. The following routine restores the status of the accumulator and the carry flags, which were previously saved by the above program segment.

Instructions	Comments
MOV A,R7	;restore carry flags status to
ADD A,#0AAH	;accumulator and set/clear
	;carry flags
MOV A,R6	;restore accumulator
RETI	;return from interrupt

An interrupt or CALL to a subroutine causes the contents of the program counter to be stored in one of the eight register pairs of the Program Counter Stack. During a CALL instruction the program counter, when saved, points to the second byte of the CALL instruction (or the return address minus one). The stack contents are then incremented before being loaded into the program counter during a return (RET) from subroutine. During an interrupt the program counter, when saved, points directly to the return address. Thus, during a return (RETI) from interrupt, the stack contents are not incremented but loaded directly into the program counter. This difference makes it imperative to use only RETI's to return from interrupts, and RET's to return from subroutines.

The interrupt system is single level in that once an interrupt is detected all further interrupt requests are ignored until execution of a RETI re-enables the interrupt input logic. This sequence holds true also for an internal interrupt generated by timer overflow. If an external interrupt and an internal timer/counter generated interrupt are detected at the same time, the external source will be recognized first, if enabled. The timer/counter interrupt will be recognized, if enabled, after the return (RETI) from the external interrupt. Timer/counter generated internal interrupts and T0 generated external interrupts have separate vector locations. The external interrupt will vector to location 3, whereas an internal interrupt will vector to location 7.

If needed, a second external interrupt can be created by enabling the timer/counter interrupt (EN TCNTI), loading FFH into the counter (one less than terminal count) and enabling the event counter mode (STRT CNT). A low-to-high transition on the T1 input will then cause an interrupt vector to location 7.

Zero Cross Detect

The Test 1 pin, in addition to being a testable input and a counter input, also serves one other important function. It can be used to detect the zero crossing point of slow moving AC signals. Execution of the STRT CNT instruction puts the T1 pin in the counter input mode by connecting T1 to the counter and enabling the counter. Subsequent low-to-high transitions on T1 will cause the counter to increment. Note that this operation differs from the rest of the MCS-48 devices, which increment the counter on high-to-low transitions. This change was made on the 8022 to take advantage of the accuracy of the rising edge detection on the zero cross circuitry.

When driven directly, this pin responds as a normal digital input. To utilize the zero cross detection mode, an AC signal of approximately 1-3 VAC p-p magnitude and a maximum frequency of 1kHz is coupled through an exter-

nal capacitor (1 microfarad) to the T1 pin. The internal digital state is sensed as a zero until the rising edge crosses the DC average level, when it becomes a one. This is accomplished by the self-biasing high gain amplifier which is included in the T1 input. This circuit biases the T1 input exactly at its switching point, such that a small change will cause a digital transition to occur. This digital transition takes place within 5 degrees of the zero point.

The digital value of T1 remains a one until the falling edge of the AC input drops approximately 100mV below the switching point of the rising edge (100mV below the zero point, if the digital transition occurred exactly at the zero point). The 100 mV offset is created by hysteresis and eliminates chattering of the internal signal caused by external noise.

The accuracy of the zero crossing will be a function of the capacitor used (see Fig. 10). On critical systems the capacitor can be adjusted to improve overall accuracy.

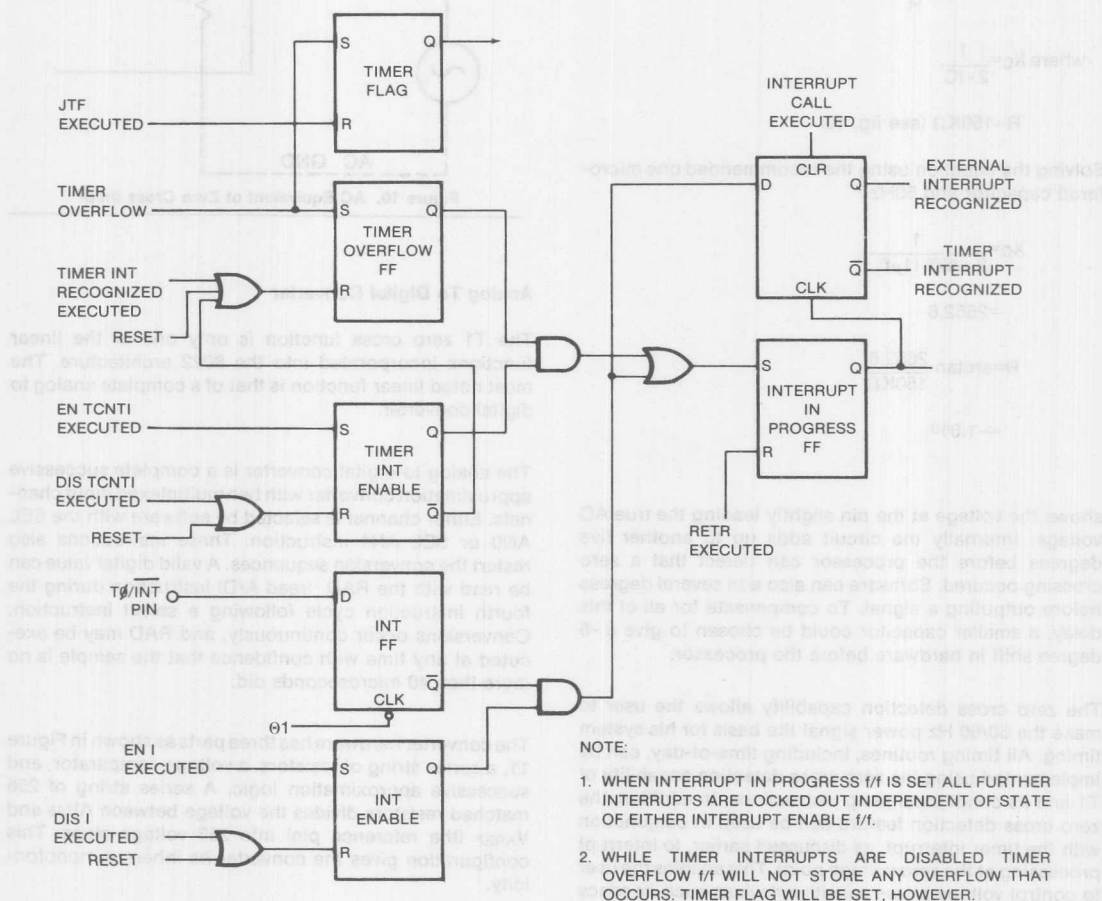


Figure 8. Interrupt Logic

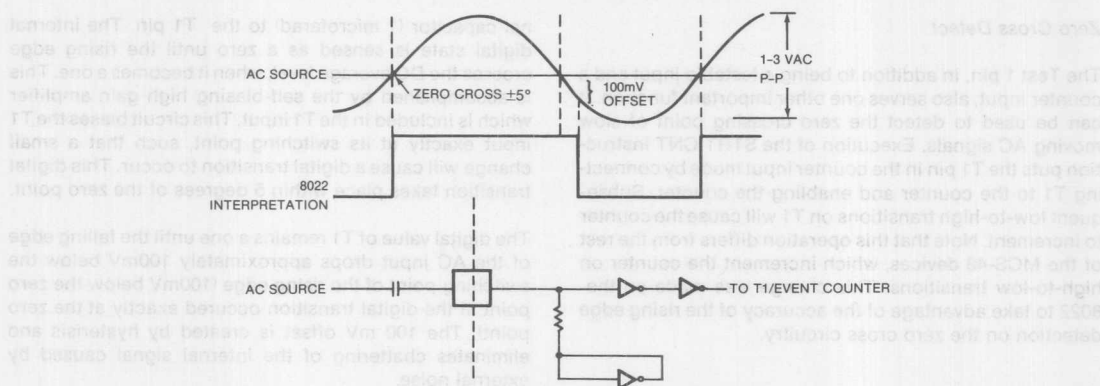


Figure 9. Zero Cross Detection

The phase angle at the T1 input can be expressed as

$$\theta = \arctan \frac{X_C}{R}$$

$$\text{where } X_C = \frac{1}{2\pi fC}$$

$$R = 150K\Omega \text{ (see fig. 10)}$$

Solving the equation using the recommended one micro-farad capacitor and 60Hz

$$X_C = \frac{1}{2\pi (60) (1\mu f)}$$

$$= 2652.6$$

$$\theta = \arctan \frac{2652.6}{150K\Omega}$$

$$= -1.010$$

shows the voltage at the pin slightly leading the true AC voltage. Internally the circuit adds up to another five degrees before the processor can detect that a zero crossing occurred. Software can also add several degrees before outputting a signal. To compensate for all of this delay, a smaller capacitor could be chosen to give a -5 degree shift in hardware before the processor.

The zero cross detection capability allows the user to make the 50/60 Hz power signal the basis for his system timing. All timing routines, including time-of-day, can be implemented using the zero cross detection capability of T1 and its conditional jump instructions. In addition, the zero cross detection feature can be used in conjunction with the timer interrupt, as discussed earlier, to interrupt processing at the zero voltage point. This enables the user to control voltage phase sensitive devices such as triacs and SCRs, and to use the 8022 in applications such as shaft speed and angle measurement.

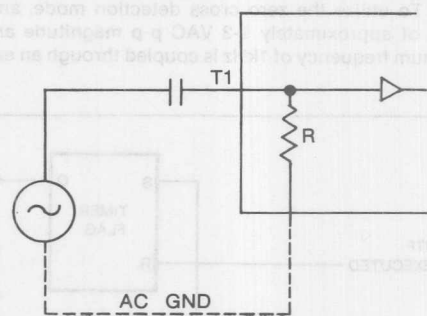


Figure 10. AC Equivalent of Zero Cross Input

Analog To Digital Converter

The T1 zero cross function is only one of the linear functions incorporated into the 8022 architecture. The most noted linear function is that of a complete analog to digital converter.

The analog to digital converter is a complete successive approximation converter with two multiplexed input channels. Either channel is selected by software with the SEL AN0 or SEL AN1 instruction. These instructions also restart the conversion sequences. A valid digital value can be read with the RAD (read A/D) instruction during the fourth instruction cycle following a select instruction. Conversions occur continuously, and RAD may be executed at any time with confidence that the sample is no more than 40 microseconds old.

The converter hardware has three parts as shown in Figure 11, a series string of resistors, a voltage comparator, and successive approximation logic. A series string of 256 matched resistors divides the voltage between AVss and VAREF (the reference pin) into 256 voltage steps. This configuration gives the converter its inherent monotonicity.

The voltage tap on the series resistor string is selected by

the resistor ladder decoder. This decoder is driven by the 8-bit successive approximation register (SAR). Each bit of the SAR is set in succession MSB to LSB and a voltage comparison between the selected resistor ladder voltage and the analog input voltage is performed after the setting of each bit. The result of each comparison determines whether the particular bit will remain set or be reset. All

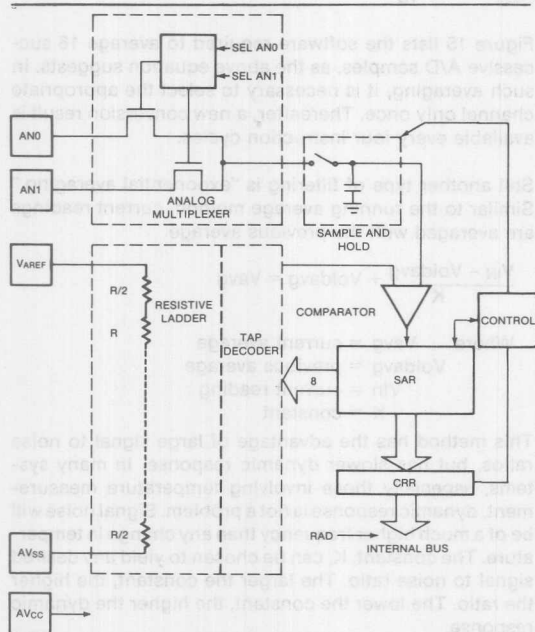


Figure 11. Analog to Digital Converter Block Diagram

comparisons are performed automatically by the on-chip A/D hardware. At the end of eight comparisons the SAR contains a valid digital representation of the analog voltage. This result is then latched into the conversion result register (CRR). The RAD instruction can then load the conversion result from the CRR to the accumulator.

To insure maximum accuracy from the A/D converter, separate power supply pins (Avcc and Avss) and a substrate pin (SUBST) have been provided. Unless there is excessive noise on the digital power supply, both Vcc and Avcc can be tied together and still maintain maximum accuracy. Figure 12 shows a typical analog configuration for sensing temperature in two thermistors. The substrate has both low frequency and high frequency bypass for noise immunity. The power supply pins (Vcc, Avcc) are bypassed with a .01 microfarad capacitor close to the chip. All other analog signals are bypassed with .001 microfarad capacitors for added noise rejection. (See also Software Noise Rejection)

As figure 11 shows, VAREF is connected to the top of the resistive ladder. When the selected analog channel is equal to or greater than VAREF the conversion result will equal 255 decimal (FF hexadecimal). The VAREF voltage can be generated in a number of ways depending on the

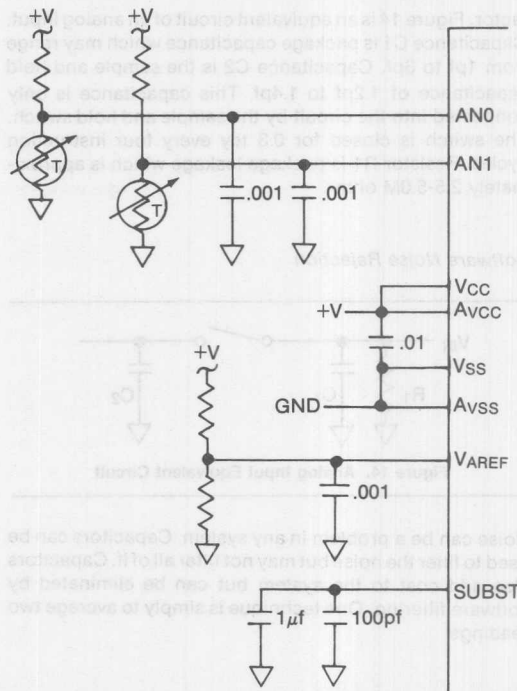


Figure 12. Typical Analog Schematic

system. It could be connected directly to Vcc giving a A/D range of GND to Vcc, or a simple resistor divider could be used to balance the reference voltage with the analog signals as in Figure 12. In calculating the impedance of the divider, the ladder impedance must be considered (see Figure 13). The total impedance of the ladder ranges from approximately 15K to 20K. This includes part to part differences and variance as a function of temperature. The resistor impedance should be chosen such that the 15K ohm parallel resistance is a small percentage of the divider impedance.

Input impedance of the converter can also be an important

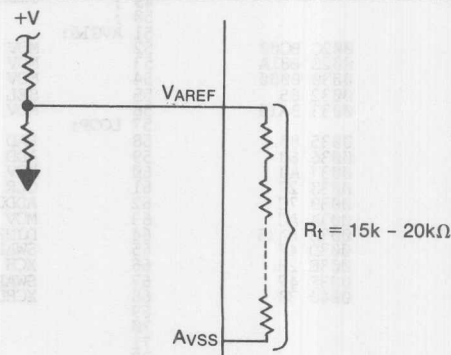


Figure 13. Ladder Impedance

factor. Figure 14 is an equivalent circuit of an analog input. Capacitance C1 is package capacitance which may range from 1pf to 3pf. Capacitance C2 is the sample and hold capacitance of 1.2pf to 1.4pf. This capacitance is only connected into the circuit by the sample and hold switch. The switch is closed for 0.3 tcy every four instruction cycles. Resistor R1 is package leakage which is approximately 2.5-5.0M ohms.

Software Noise Rejection

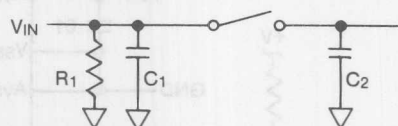


Figure 14. Analog Input Equivalent Circuit

Noise can be a problem in any system. Capacitors can be used to filter the noise but may not filter all of it. Capacitors also add cost to the system but can be eliminated by software filtering. One technique is simply to average two readings:

$$\frac{VIN1 + VIN2}{2} = VOUT$$

or keep a running average by averaging each reading with the previous average:

```
SEL AN0      ;Start conversion
MOV R0,#30   ;Point to storage location
RAD          ;Read current A/D sample
ADD A,@R0    ;Add current sample to previous average
RRC A        ;Divide by two
MOV @R0,A    ;Store new average
```

This method will eliminate small fluctuations in the input voltage and reduce the effect of large fluctuations. Often, however, noise may be more severe. Excessive noise may require averaging of many readings taken over a short period of time.

$$\frac{VIN1 + VIN2 + \dots + VIN16}{16} = VOUT$$

Figure 15 lists the software required to average 16 successive A/D samples, as the above equation suggests. In such averaging, it is necessary to select the appropriate channel only once. Thereafter, a new conversion result is available every four instruction cycles.

Still another type of filtering is "exponential averaging." Similar to the running average method, current readings are averaged with the previous average.

$$\frac{VIN - Voldavg}{K} + Voldavg = Vavg$$

Where Vavg = current average
Voldavg = previous average
Vin = current reading
K = constant

This method has the advantage of large signal to noise ratios, but has slower dynamic response. In many systems, especially those involving temperature measurement, dynamic response is not a problem. Signal noise will be of a much higher frequency than any change in temperature. The constant, K, can be chosen to yield any desired signal to noise ratio. The larger the constant, the higher the ratio. The lower the constant, the higher the dynamic response.

To increase the effectiveness in reducing line generated noise, any of the above methods should be synchronized to the line frequency. As previously discussed, an interrupt can be generated when the 50Hz or 60Hz line frequency crosses AC zero. The A/D filtering routine should be part of the interrupt routine. Reading of the A/D will then occur at the same point of each line cycle, thus ignoring any line generated fluctuations in the analog inputs.

LOC	OBJ	LINE	SOURCE STATEMENT
		47 ;	
		48 ;	
		49 ;	AVERAGE 16 A/D READINGS
		50 ;	=====
		51	AVG16:
002C	BC00	52	MOV R4,#00 ;CLEAR TEMP. MSB RESULT REGISTER
002E	B81A	53	MOV R0,#26 ;SET UP POINTER
0030	B000	54	MOV @R0,#00 ;CLEAR RESULT REGISTER
0032	85	55	SEL AN0 ;SELECT AND START CONVERSION
0033	BA10	56	MOV R2,#16 ;16 READINGS
		57	LOOP:
0035	80	58	RAD ;READ RESULT
0036	60	59	ADD A,@R0 ;LSB
0037	A0	60	MOV @R0,A ;SAVE NEW LSB
0038	27	61	CLR A
0039	7C	62	ADDC A,R4 ;ADD CARRY TO MSB
003A	AC	63	MOV R4,A
003B	EA35	64	MOV R4,A ;SAVE NEW MSB
003D	47	65	DJNZ R2,LOOP ;NEXT READING
003E	20	66	SWAP A ;MSB INTO MSN
003F	47	67	XCH A,@R0
0040	30	68	SWAP A ;DIVIDE BY 16
		69	XCHD A,@R0 ;LOCATION 26 NOW CONTAINS
		70	;THE AVERAGE
		71	
		72	

Figure 15.

Port 0 Comparator Inputs

Intel, in its commitment to add analog features to microcomputers, did not stop with A/D conversion and zero cross detection. Also added to the 8022 were eight comparators for easing the interface to non-digital inputs.

Port 0 has been modified from the standard quasi-bidirectional structure to allow an optional open drain configuration with comparator inputs. The low impedance pullup device has been eliminated and the high impedance pullup is optional. Thus, the user can choose via a mask programmable selection each line of Port 0 to be either quasi-bidirectional with a high impedance or true open-drain. The open drain configuration allows the line to sink current through the low impedance pulldown device or to float in the high output state. More importantly, the open drain configuration makes Port 0 very easy to drive when it is used as inputs. The input circuitry for each line of Port 0 includes a voltage comparator which amplifies the voltage difference between the input port line and the Port 0 threshold reference pin (V_{TH}). The voltage gain of the comparator is sufficient to sense a 100mV input differential within the range V_{SS} to $V_{CC}/2$.

If V_{TH} is allowed to float, it will bias itself to the digital switch point of the other ports, and Port 0 behaves as a set of normal digital inputs. However, by biasing V_{TH} , the switch point can be both tightly controlled and adjusted.

Common uses for this would include unusual logic level inputs as from a diode isolated keyboard, analog channel expansion, and direct capacitive touchpanel interface. The comparator action is automatic and the port is read just as any other port.

A typical use for Port 0 is in the interfacing with capacitive touch panels on microwave ovens and other new appliances. A touch-panel switch consists of two capacitors in series. One lead is attached to a high voltage buffer (10 to 30 volts). The other is attached to the Port 0 sense input. As a finger touches the common point, the drive signal is shunted by body capacitance, attenuating the signal reaching the input.

Low-voltage touch-panel operation (less than 30V) is possible, since the comparators allow small voltage changes to be detected. Most of the present touch-panel designs require a 50-100V drive on the touch panel.

Capacitive touch panels can be multiplexed in the same manner as mechanical keyboards (Fig. 16). The vacuum fluorescent display and the touch panel drivers are integrated to optimize hardware through shared high voltage buffers.

Figure 17 lists the software necessary to refresh the display and scan the touch-panel matrix. This routine could be adapted to serve as part of a timer/interrupt

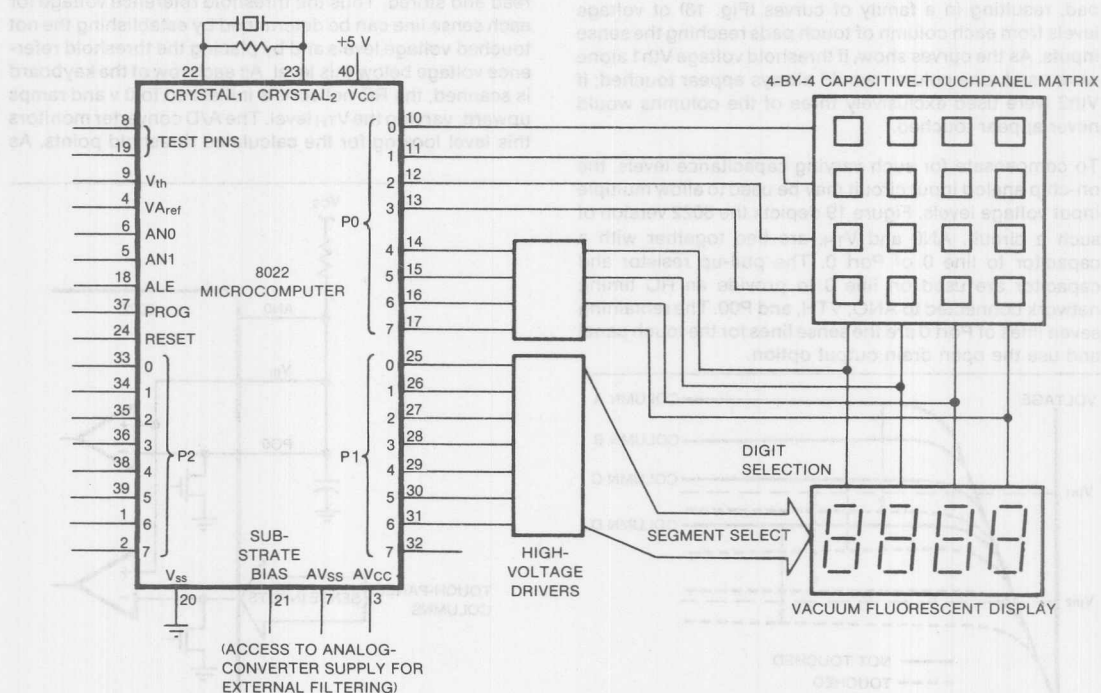


Figure 16. Typical Keyboard/Display Schematic

LOC	OBJ	LINE	SOURCE STATEMENT
		74 ;	
		75 ;	KEYBOARD DISPLAY ROUTINE
		76 ;	=====
		77 ;	
003C		78 D4 EQU 3CH ;MSD OF DISPLAY	
		79 KEYDIS:	
0041 27		80 CLR A	
0042 39		81 OUTL P1,A ;TURN OFF SEGMENT DRIVERS	
0043 90		82 OUTL P0,A ;TURN OFF DIGIT DRIVERS AND	
		83 ;PANEL STROBES	
		84 ;INITIALIZE SENSE INPUTS TO GND	
0044 230F		85 MOV A,#0FH	
0046 90		86 OUTL P0,A ;FLOAT SENSE INPUTS	
0047 4B		87 ORL A,R3 ;NEW STROBE POSITION	
0048 90		88 OUTL P0,A ;TURN ON STROBE	
0049 08		89 IN A,P0 ;READ SENSE INPUTS	
004A AC		90 MOV R4,A ;SAVE SENSE INPUTS	
004B B83B		91 MOV R0,#D4-1 ;RAM LOCATION OF MSB OF 7-SEG PATTERN	
004D FB		92 MOV A,R3 ;STROBE POSITION INTO A	
004E 90		93 OUTL P0,A ;GND SENSE INPUTS	
		94 LOOP1:	
004F F7		95 RLC A ;ROTATE DIGIT STROBE INTO CARRY	
0050 18		96 INC R0 ;NEXT DIGIT LOCATION	
0051 E64F		97 JNC LOOP1 ;LOOP UNTIL CARRY	
0053 F0		98 MOV A,@R0 ;RETRIEVE PATTERN FROM RAM	
0054 39		99 OUTL P1,A ;OUTPUT NEW PATTERN	
		100	
		101	

Figure 17.

scheme that would generate an interrupt at precise intervals for a flicker-free display. Another portion of the software would check for any touched input pads, test for valid entry, and enter key-depression codes into the main program.

Correcting Pad Imbalance

A common problem with capacitive touch panels is their imbalance. Layout, process, aging, and surface impurities all cause the capacitance to vary from touch pad to touch pad, resulting in a family of curves (Fig. 18) of voltage levels from each column of touch pads reaching the sense inputs. As the curves show, if threshold voltage V_{th1} alone were used, one column would always appear touched; if V_{th2} were used exclusively three of the columns would never appear touched.

To compensate for such varying capacitance levels, the on-chip analog input circuit may be used to allow multiple input voltage levels. Figure 19 depicts the 8022 version of such a circuit. $AN0$ and V_{th} are tied together with a capacitor to line 0 of Port 0. The pull-up resistor and capacitor are used on line 0 to provide an RC timing network connected to $AN0$, V_{th} , and $P00$. The remaining seven lines of Port 0 are the sense lines for the touch panel and use the open drain output option.

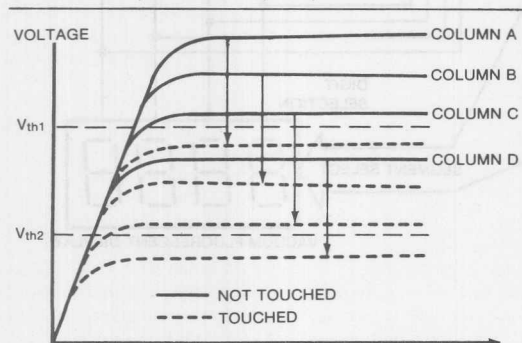


Figure 18.

Handling the different voltage levels would begin with initializing the data by plotting the family of curves with the $AN0$ input. This is done by writing a 0 to $P00$ (grounding $P00$) which initializes V_{th} to 0v. A logic 1 is then written to $P00$, which begins to pull the RC network toward 5 v. As V_{th} ramps upward, the sense lines are monitored for input changes, which will go from 1 to 0 as the not-touched voltage curve for each input is intersected. As the changes occur, the A/D value for each input is intersected. As the changes occur, the A/D value for each sense line can be read and stored. Thus the threshold reference voltage for each sense line can be determined by establishing the not touched voltage levels and by placing the threshold reference voltage below this level. As each row of the keyboard is scanned, the RC network is initialized to 0 v and ramps upward, varying the V_{th} level. The A/D converter monitors this level looking for the calculated threshold points. As

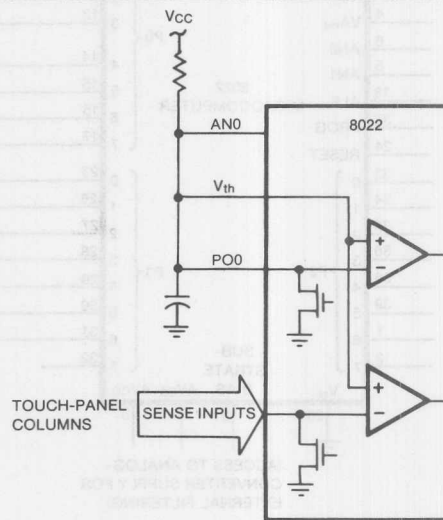


Figure 19.

the points are intersected, the corresponding sense inputs are read, with a 0 indicating a touched input and a 1 indicating a not-touched input. Thus, a multiplexed capacitive touch panel can be scanned and balanced without adding external components to the inputs.

For systems requiring more than two analog inputs to the 8022, Port 0 comparator inputs may be reconfigured to permit formation of pseudo-analog inputs from variable threshold digital inputs. The hardware configuration can be identical to that of Fig. 19. In this scheme, sense inputs act as additional analog inputs of less accuracy than AN0 and AN1 (about 6 bits).

The sequence for implementing the extension is essentially the same found in the variable-threshold touch panel. As V_{TH} ramps upward, a Port 0 bit is monitored for a change from 1 to 0. At the change, AN0 is read corresponding to the value of the analog input into Port 0 (with some error due to the time lag, which can be subtracted). This configuration can be utilized when it is possible to trade off accuracy for such cost improvements: one analog input with 8-bit accuracy and seven analog inputs with 6-bit accuracy. Such may be the case in a range controller that monitors temperature in two ovens, a meat probe, and two of the four burners, all to an accuracy within 10°F.

Application Ideas

This section will discuss some possible applications of the 8022. These applications are discussed in general terms and are believed to be feasible applications of the 8022. None of these applications, however, have been built and checked out.

Power Supply Controller

The three terminal voltage regulator, with its built-in current limiting and overload protection, has vastly simplified the task of designing small power supplies. Power supplies for large systems, with requirements for brown out protection, power fail warnings, etc., have not yet yielded to the design simplicity of the integrated voltage regulator. The combination of an 8022 microcomputer and these same regulators, however, may make it feasible to simplify these larger power supply systems.

There are several requirements of larger power supplies which have to be met outside of the regulation itself. Typical of these are:

1. Sequencing the turn on and shut down of several supplies.
2. Providing an early warning to the system that power is failing.
3. The ability to hold the system in a reset state during power supply sequencing.
4. Generation of a line frequency clock to the system.
5. Provisions for remote start up and shut down.
6. Sufficient energy storage to keep the system running long enough to provide an orderly shut down.
7. High efficiencies to minimize power requirements and heat dissipation.

These requirements can be met by a combination of raw DC supply, multiple three-terminal regulators, and an 8022 microcomputer. Figure 20 shows a raw supply which is capable of generating DC voltages suitable for regulation to five, plus twelve, and minus twelve voltages. (These are arbitrary, but common voltages). In addition, a separate winding is provided which generates a five-volt supply which will be used to supply power to the 8022 itself. The normal rectifiers in the RAW5 and RAW12 supplies are replaced by silicon controlled rectifiers which will be phase angle controlled by the 8022.

Figure 21 shows the connections to the 8022. The RAW5 and RAW12 supplies are applied to simple voltage dividers which feed the analog inputs of the 8022. The signal

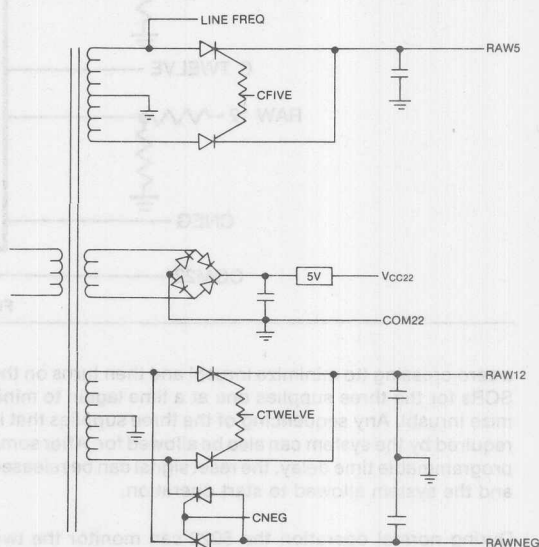


Figure 20.

LINEFREQ is taken from a convenient winding of a transformer, divided down, and applied to the zero cross input of the 8022. A strap is provided to configure the unit for 50/60 Hz operation. In addition to being connected to the basic power supply, the 8022 is also connected to the system receiving the power. The on/off switch becomes an input to the 8022. The 8022 provides outputs for a 10Hz interrupt, a power fail interrupt, cold/warm indicator and system reset.

The 8022 can perform many of the functions normally done by hardware sequencers in the power supply. On power-up, it can hold off the three main supplies until the main supply is firmly established. This prevents the system from responding to short power restorations which frequently occur during power outages. Having determined that it is safe to power up the system, the 8022 can assert the reset signal and the cold start signal. The cold start indication tells the system that power was interrupted at the mains rather than by the OFF switch—a useful function if any amount of battery backed up RAM exists in the system. Having set up these signals, the 8022 waits for

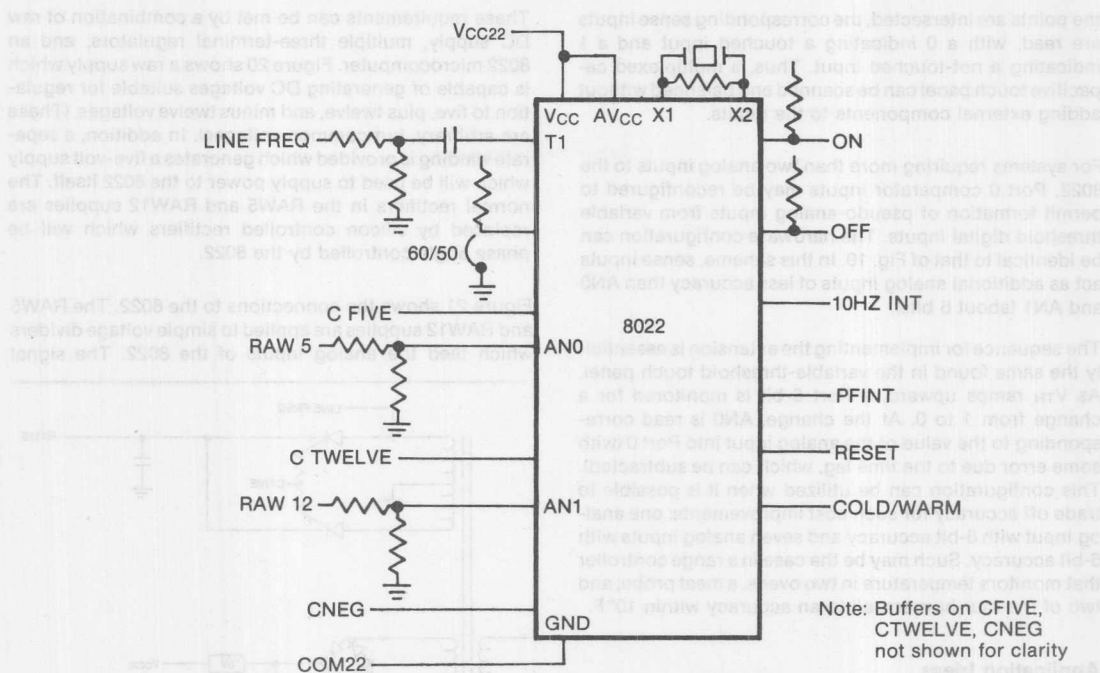


Figure 21.

a zero crossing (to minimize inrush) and then turns on the SCRs for the three supplies one at a time (again to minimize inrush). Any sequencing of the three supplies that is required by the system can also be allowed for. After some programmable time delay, the reset signal can be released and the system allowed to start operation.

During normal operation the 8022 can monitor the two major raw supplies and use phase angle control of the SCRs to regulate them. The regulation would be used to ensure that the three terminal regulators had minimum input voltage requirements met under all line voltage variations while at the same time minimizing the voltage drop across them. This increases the efficiency of the power supply and allows it to be capable of handling brown outs without dissipating excessive power in the regulators.

The line frequency input is used not only for the basis for the phase angle control, but also for two other functions; power fail detect and generation of the 10 Hz interrupt. The 10Hz interrupt can be generated by simply dividing the power line frequency by 5 for 50 Hz and 6 for 60 Hz operation. Performing this division in the power supply itself allows the system to be run on 50 or 60 cycle power with no change external to the power supply. In some situations it should even be possible to have the power supply adapt to either of these inputs by measuring the period of the incoming power on startup (see section "Which One?"). This would be an easy function to incorporate in the software and would require no additional hardware since provision is already made for zero cross detect.

Power fail detection can be done by running the timer while waiting for the line to zero cross. If an excessive time elapses it can be assumed that the power has failed and the power fail interrupt asserted. Note that this will detect total power failure but not a dip in the line voltage below the specifications of the power supply. This condition can be detected by keeping track of the phase angle that is required to maintain the RAW supplies at the proper level. If the SCR's have to be turned on for too high a portion of the total line cycle it is an indication of a brown-out condition and the powerfail interrupt should be generated. Whenever the powerfail interrupt is generated the 8022 should turn on the SCRs continuously to ensure maximum possible energy storage in the filter capacitors. After generation of the powerfail interrupt, the 8022 can again delay (depending, of course, on the energy storage of the power supply) and then assert reset. Once reset is asserted the SCRs are turned off, and left off, until the supplies have dropped down to a point which guarantees that any reset circuitry residing outside of the power supply will see a full power transition when power is reapplied. If the power is shut down by the 8022 in response to the on/off switch, the sequence would be similar except that the cold/warm start signal would indicate a warm start.

The above discussion should make it clear that the 8022 would make the task of designing a power supply system far easier, particularly for those designers more familiar with digital than analog design. If, in addition, the 8022 supply were put on a battery back-up, it would be possible to add many features to the system at virtually zero cost. The 8022 could be programmed to become the system clock and send, perhaps in serial ASCII, the time of day

and the date to the main system on demand or periodically. This function would require that a crystal be used as a timing reference to the 8022 so that the power supply could still track real time even if the incoming power fails. Other possibilities would have the system shut down unless some external event required its attention, or the incorporation of system diagnostic checks within the code of the 8022. The comparator inputs on PORT 0 of the 8022 would even allow some capability of parametric testing as part of these diagnostics. The possibilities bring a new dimension to the term "Programmable Power Supply".

DC Motor Control

Figure 22 shows the 8022 used to control the speed of a permanent magnet DC motor. A seven segment display and keyboard are provided which allow the user to enter the parameters required by the control algorithm. The display is also used to display the speed of the motor

during operation. Other data (for example root mean squared error) could also be displayed upon demand. The motor is driven by a constant frequency pulse width modulated signal which is generated programmatically. Port 11 (which is one of the two high current outputs) is used to drive a photoisolator which provides level shifting as well as isolation. The circuit shown allows both the speed and torque of the motor to be measured for use by the control algorithm. The torque generated by a PM DC motor is proportional to the armature current. This current, and hence the torque, can be measured by reading the voltage drop across the shunt resistor. The voltage generated across the motor is the sum of the IR drop in the armature and a term which is proportional to the angular speed of the motor. The armature current is already known from the torque measurement, so the speed can easily be determined from the two analog measurements shown. The DC resistance of the armature, the speed constant, and torque constant would, of course, have to be known or entered by the operator.

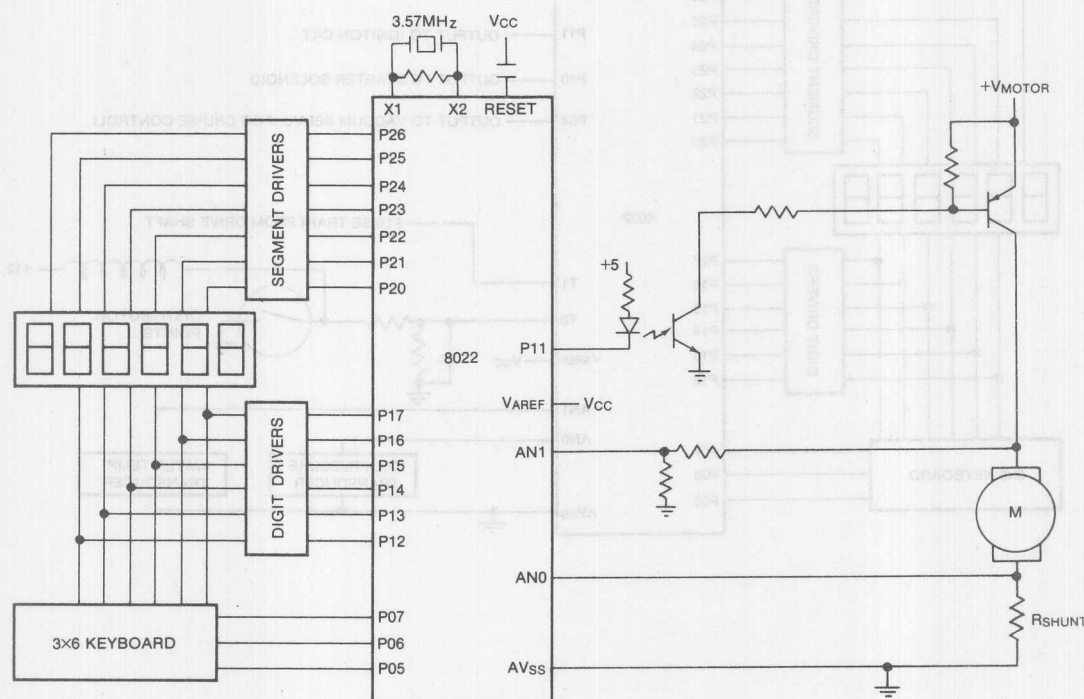


Figure 22. DC Motor Control

distributor points allows the engine RPM and point dwell to be measured. Outputs are provided to control the ignition and starter (allowing the ignition switch to be eliminated in favor of a combination lock). Drive to a

speedometer, tachometer, or pressure gauge, or water temperature gauge depending on the current desire of the driver. There are several uncommitted I/O pins which could be used to implement functions such as intermittent action windshield wipers or delayed action light circuits.

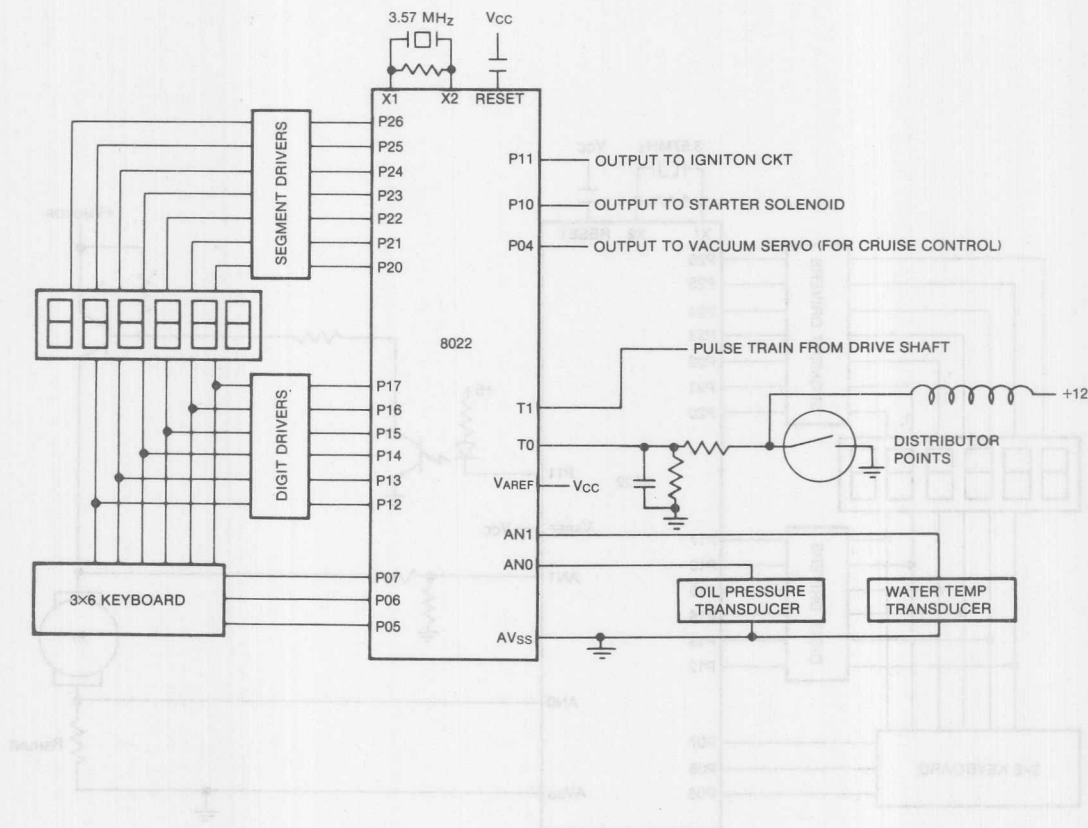


Figure 23. Automotive Dashboard

Darkroom Timer

A darkroom timer based on the 8022 is shown in Figure 24. In addition to the keyboard and display this diagram incorporates drive to two TRIACs, an input to monitor the line frequency crossings, and two analog measurements. The analog inputs are used to monitor and display the temperature of the chemical bath and the light output of

the enlarger, both of which can be controlled by the microcomputer. The 8022 could be used to run several timers concurrently while also maintaining the temperature of the chemical bath at the required level. Several uncommitted I/O pins are available for additional functions.

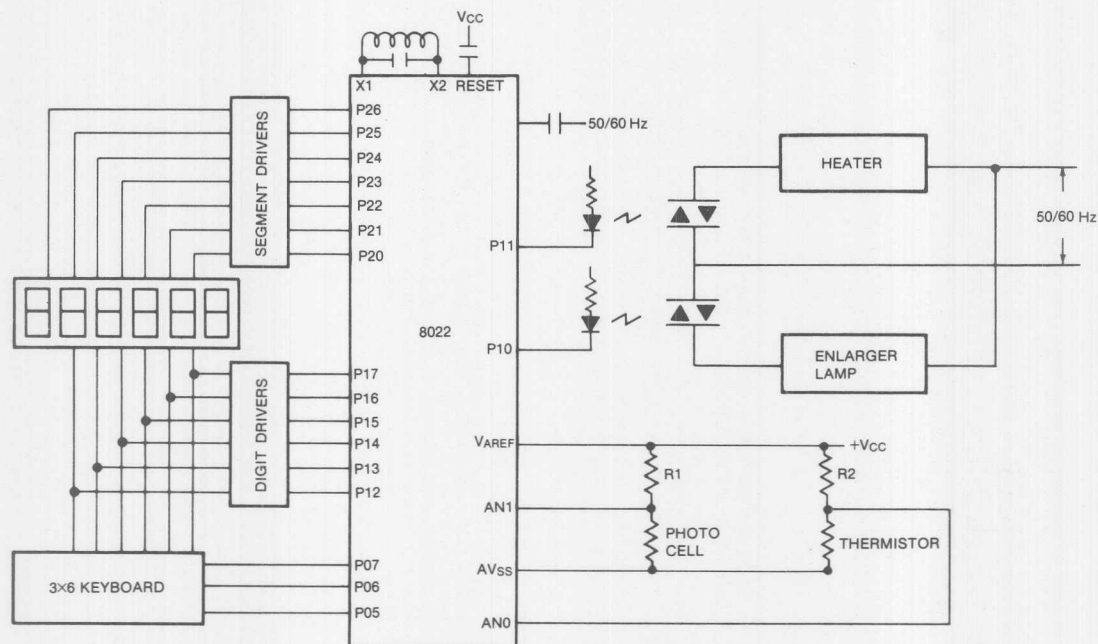


Figure 24. Darkroom Timer/Control

Conclusions

This application note has introduced the reader to the Intel 8022 microcomputer. It has described the main features of the 8022 and discussed some of the design considerations

encountered in designing with the 8022.

The reader has also been exposed to several possible applications which show the versatility and cost effectiveness of a microcomputer with on-board analog features.

Contents	
Introduction	9-3
UPI Master Protocol	9-3
Example Applications	9-9
8-Digit Multiplexed LED Display Controller	9-11
8-Digit Matrix Controller	9-14
Combination I/O Device	9-18
Design Techniques	9-23
Appendix A1	9-27
Appendix A2	9-31
Appendix B1	9-33
Appendix B2	9-44
Appendix C1	9-46
Appendix C2	9-51
Conclusion	9-58

November 1978

Introduction to the UPI-41A

John Beaston and Robin Jigour
Microcomputer Applications

Introduction to the UPI-41A

Contents

Introduction	9-3
UPI/Master Protocol	9-3
Example Applications	9-9
8-Digit Multiplexed LED Display Controller	9-11
Sensor Matrix Controller	9-14
Combination I/O Device	9-18
Debug Techniques	9-25
Appendix A1	9-27
Appendix A2	9-34
Appendix B1	9-36
Appendix B2	9-44
Appendix C1	9-46
Appendix C2	9-61
Conclusion	9-26

INTRODUCTION

Since the introduction in 1974 of the second generation of microprocessors, such as the 8080, a wide range of peripheral interface devices have appeared. At first, these devices solved application problems of a general nature; i.e., parallel interface (8255), serial interface (8251), timing (8253), interrupt control (8259). However, as the speed and density of LSI technology increased, more and more intelligence was incorporated into the peripheral devices. This allowed more specific application problems to be solved, such as floppy disk control (8271), CRT control (8275), and data link control (8273). The advantage to the system designer of this increased peripheral device intelligence is that many of the peripheral control tasks are now handled externally to the main processor in the peripheral hardware rather than internally in the main processor software. This reduced main processor overhead results in increased system throughput and reduced software complexity.

In spite of the number of peripheral devices available, the pervasiveness of the microprocessor has been such that there is still a large number of peripheral control applications not yet satisfied by dedicated LSI. Complicating this problem is the fact that new applications are emerging faster than the manufacturers can react in developing new, dedicated peripheral controllers. To address this problem, a new microcomputer-based Uni-

versal Peripheral Interface (UPI-41A) device was developed.

In essence, the UPI-41A acts as a slave processor to the main system CPU. The UPI contains its own processor, memory, and I/O, and is completely user programmable; that is, the entire peripheral control algorithm can be programmed locally in the UPI, instead of taxing the master processor's main memory. This distributed processing concept allows the UPI to handle the real-time tasks such as encoding keyboards, controlling printers, or multiplexing displays, while the main processor is handling non-real-time dependent tasks such as buffer management or arithmetic. The UPI relies on the master only for initialization, elementary commands, and data transfers. This technique results in an overall increase in system efficiency since both processors — the master CPU and the slave UPI — are working in parallel.

This application note presents three UPI-41A applications which are roughly divided into two groups: applications whose complexity and UPI code space requirements allow them to either stand alone or be incorporated as just one task in a "multi-tasking" UPI, and applications which are complete UPI applications in themselves. Applications in the first group are a simple LED display and sensor matrix controllers. A combination serial/parallel I/O device is an application in the second group. Each application illustrates different UPI config-

UPI-41 vs. UPI-41A

The UPI-41A is an enhanced version of the UPI-41. It incorporates several architectural features not found on the "non-A" device:

- Separate Data In and Data Out data bus buffer registers
- User-definable STATUS register bits
- Programmable master interrupts for the OBF and $\overline{\text{IBF}}$ flags
- Programmable DMA interface to external DMA controller.

The separate Data In (DBBIN) and Data Out (DBBOUT) registers greatly simplify the master/UPI protocol compared to the UPI-41. The master need only check IBF before writing to DBBIN and OBF before reading DBBOUT. No data bus buffer lock-out is required.

The most significant nibble of the STATUS register, undefined in the UPI-41, is user-definable in UPI-41A. It may be loaded directly from the most significant nibble of the Accumulator (MOV STS, A). These extra four STATUS bits are useful for transferring additional status information to the master. This application note uses this feature extensively.

A new instruction, EN FLAGS, allows OBF and $\overline{\text{IBF}}$ to be reflected on Port 2 bit 4 and Port 2 bit 5 respectively. This feature enables interrupt-driven data transfers when these pins are interrupt sources to the master.

By executing an EN DMA instruction Port 2 bit 6 becomes a DRQ (DMA Request) output and Port 2 bit 7 becomes $\overline{\text{DACK}}$ (DMA Acknowledge). Setting DRQ requests a DMA cycle to an external DMA controller. When the cycle is granted, the DMA controller returns $\overline{\text{DACK}}$ plus either RD (Read) or WR (Write). $\overline{\text{DACK}}$ automatically forces $\overline{\text{CS}}$ and A0 low internally and clears DRQ. This selects the appropriate data buffer register (DBBOUT for $\overline{\text{DACK}}$ and RD, DBBIN for $\overline{\text{DACK}}$ and WR) for the DMA transfer.

Like the "non-A", the UPI-41A is available in both ROM (8041A) and EPROM (8741A) Program Memory versions. This application note deals exclusively with the UPI-41A since the applications use the "A"'s enhanced features.

urations and features. However, before the application details are presented, a section on the UPI/master protocol requirements is included. These protocol requirements are key to UPI software development. It is suggested that the reader not already familiar with the

architecture and instruction set of the UPI-41A read the "Intel UPI-41 User's Manual" before proceeding with this document. For convenience, the UPI block diagram and instruction set summary are reproduced in Figures 1 and 2.

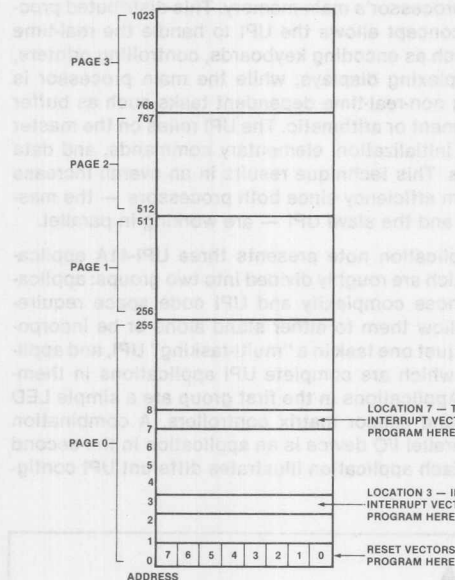


Figure 1A. Program Memory Map

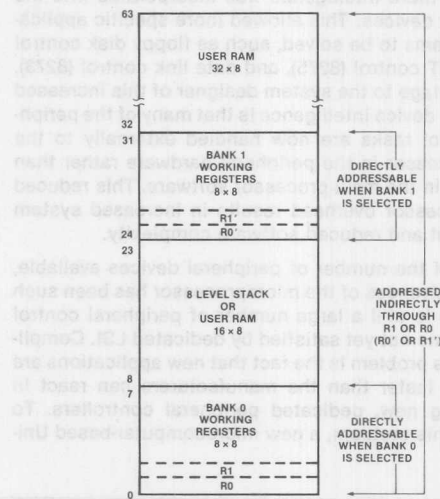


Figure 1B. Data Memory Map

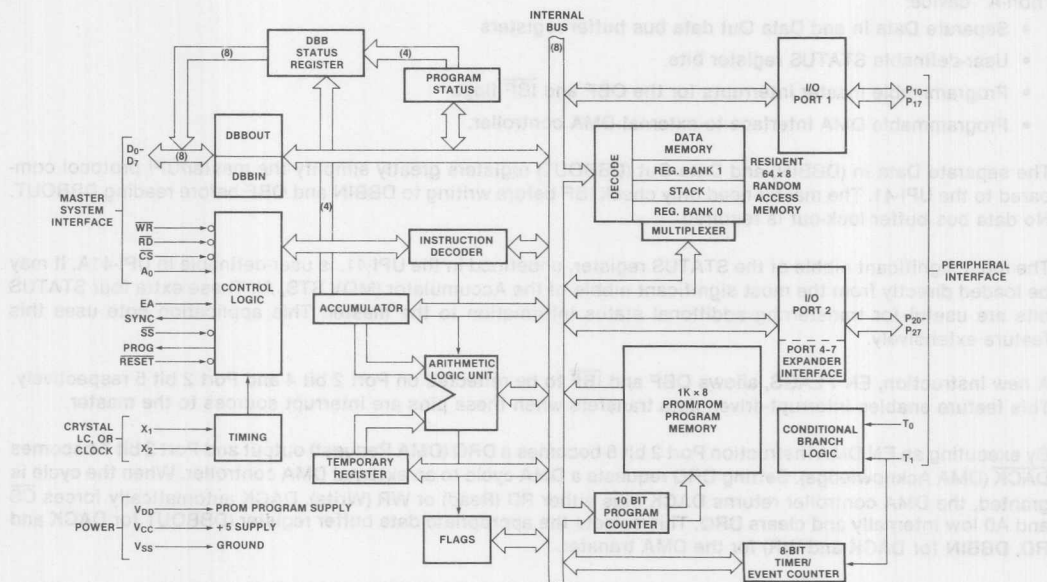


Figure 1C. UPI-41A Block Diagram

UPI INSTRUCTION SET

Mnemonic	Description	Bytes	Cycles
ACCUMULATOR			
ADD A,Rr	Add register to A	1	1
ADD A,@Rr	Add data memory to A	1	1
ADD A,#data	Add immediate to A	2	2
ADDC A,Rr	Add immed. to A with carry	1	1
ADDC A,@Rr	Add immed. to A with carry	1	1
ADDC A,#data	Add immed. to A with carry	2	2
ANL A,Rr	AND register to A	1	1
ANL A,@Rr	AND data memory to A	1	1
ANL A,#data	AND immediate to A	2	2
ORL A,Rr	OR register to A	1	1
ORL A,@Rr	OR data memory to A	1	1
ORL A,#data	OR immediate to A	2	2
XRL A,Rr	Exclusive OR register to A	1	1
XRL A,@Rr	Exclusive OR data memory to A	1	1
XRL A,#data	Exclusive OR immediate to A	2	2
INC A	Increment A	1	1
DEC A	Decrement A	1	1
CLR A	Clear A	1	1
CPL A	Complement A	1	1
DA A	Decimal Adjust A	1	1
SWAP A	Swap digits of A	1	1
RL A	Rotate A left	1	1
RLC A	Rotate A left through carry	1	1
RR A	Rotate A right	1	1
RRC A	Rotate A right through carry	1	1

INPUT/OUTPUT

IN A,Pp	Input port to A	1	2
OUTL Pp,A	Output A to port	1	2
ANL Pp,#data	AND immediate to port	2	2

Mnemonic	Description	Bytes	Cycles
CONTROL			
EN DMA	Enable DMA Handshake Lines	1	1
EN I	Enable IBF Interrupt	1	1
DIS I	Disable IBF Interrupt	1	1
EN FLAGS	Enable Master Interrupts	1	1
SEL RB0	Select register bank 0	1	1
SEL RB1	Select register bank 1	1	1
NOP	No Operation	1	1
REGISTERS			
INC Rr	Increment register	1	1
INC @Rr	Increment data memory	1	1
DEC Rr	Decrement register	1	1
SUBROUTINE			
CALL addr	Jump to subroutine	2	2
RET	Return	1	2
RETR	Return and restore status	1	2
FLAGS			
CLR C	Clear Carry	1	1
CPL C	Complement Carry	1	1
CLR F0	Clear Flag 0	1	1
CPL F0	Complement Flag 0	1	1

ORL Pp,#data	OR immediate to port	2	2
IN A,DBB	Input DBB to A, clear IBF	1	1
OUT DBB A	Output A to DBB, set OBF	1	1
MOVD A,Pp	Input Expander port to A	1	2
MOVD Pp,A	Output A to Expander port	1	2
ANLD Pp,A	AND A to Expander port	1	2
ORLD Pp,A	OR A to Expander port	1	2

DATA MOVES

MOV A,Rr	Move register to A	1	1
MOV A,@Rr	Move data memory to A	1	1
MOV A,#data	Move immediate to A	2	2
MOV Rr,A	Move A to register	1	1
MOV @Rr,A	Move A to data memory	1	1
MOV Rr,#data	Move immediate to register	2	2
MOV @Rr,#data	Move immediate to data memory	2	2
MOV A,PSW	Move PSW to A	1	1
MOV PSW,A	Move A to PSW	1	1
XCH A,Rr	Exchange A and register	1	1
XCH A,@Rr	Exchange A and data memory	1	1
XCHD A,@Rr	Exchange digit of A and register	1	1
MOVP A,@A	Move to A from current page	1	2
MOVP3 A,@A	Move to A from page 3	1	2

TIMER/COUNTER

MOV A,T	Read Timer/Counter	1	1
MOV T,A	Load Timer/Counter	1	1
STRT T	Start Timer	1	1
STRT CNT	Start Counter	1	1
STOP TCNT	Stop Timer/Counter	1	1
EN TCNTI	Enable Timer/Counter Interrupt	1	1
DIS TCNTI	Disable Timer/Counter Interrupt	1	1

Mnemonic	Description	Bytes	Cycles
CLR F1	Clear F1 Flag	1	1
CPL F1	Complement F1 Flag	1	1
MOV STS, A	A ₄ -A ₇ to Bits 4-7 of Status	1	1
BRANCH			
JMP addr	Jump unconditional	2	2
JMPP @A	Jump indirect	1	2
DJNZ R,addr	Decrement register and skip	2	2
JC addr	Jump on Carry = 1	2	2
JNC addr	Jump on Carry = 0	2	2
JZ addr	Jump on A Zero	2	2
JNZ addr	Jump on A not Zero	2	2
JT0 addr	Jump on T0 = 1	2	2
JNT0 addr	Jump on T0 = 0	2	2
JT1 addr	Jump on T1 = 1	2	2
JNT1 addr	Jump on T1 = 0	2	2
JF0 addr	Jump on F0 Flag = 1	2	2
JF1 addr	Jump on F1 Flag = 1	2	2
JTF addr	Jump on Timer Flag = 1, Clear Flag	2	2
JNIBF addr	Jump on IBF Flag = 0	2	2
JOBF addr	Jump on OBF Flag = 1	2	2
JBB addr	Jump on Accumulator Bit	2	2

Figure 2. UPI-41A Instruction Set Summary

UPI/MASTER PROTOCOL

As in most closely coupled multiprocessor systems, the various processors communicate via a shared resource. This shared resource is typically specific locations in RAM or in registers through which status and data are passed. In the case of a master processor and a UPI-41A, the shared resource is 3 separate, master-addressable, registers internal to the UPI. These registers are the STATUS register (STATUS), the Data Bus Buffer Input register (DBBIN), and the Data Bus Output register (DBBOUT). [Data Bus Buffer direction is relative to the UPI]. To illustrate this register interface, consider the 8085A/UPI system in Figure 3.

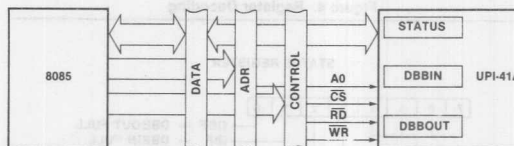


Figure 3. Register Interface

Looking into the UPI from the 8085A, the 8085A sees only the three registers mentioned above. If the 8085A wishes to issue a command to the UPI, it does so by writing the command to the DBBIN register according to the decoding of Figure 4. Data for the UPI is also passed via the DBBIN register. (The UPI differentiates commands and data by examining the A0 pin. Just how this is done is covered shortly.) Data from the UPI for the 8085A is passed in the DBBOUT register. The 8085A may interrogate the UPI's status by reading the UPI's STATUS register. Four bits of the STATUS register act as flags and are used to handshake data and commands into and out of the UPI. The STATUS register format is shown in Figure 5.

Bit 0 is OBF (Output Buffer Full). This flag indicates to the master when the UPI has placed data in the DBBOUT register. OBF is set when the UPI writes to DBBOUT and is reset when the master reads DBBOUT. The master finds meaningful data in the DBBOUT register only when OBF is set.

The Input Buffer Full (IBF) flag is bit 1. The UPI uses this flag as an indicator that the master has written to the DBBIN register. The master uses IBF to indicate when the UPI has accepted a particular command or data byte. The master should examine IBF before outputting anything to the UPI. IBF is set when the master writes to DBBIN and is reset when the UPI reads DBBIN. The master must wait until IBF = 0 before writing new data or commands to DBBIN. Conversely, the UPI must ensure IBF = 1 before reading DBBIN.

The third STATUS register bit is F0 (Flag 0). This is general purpose flag that the UPI can set, reset, and test. It is typically used to indicate a UPI error or busy condition to the master.

Flag 1 (F1) is the final dedicated STATUS bit. Like F0 the UPI can set, reset, and test this flag. However, in addition, F1 reflects the state of the A0 pin whenever the master writes to the DBBIN register. The UPI uses this flag to delineate between master command and data writes to DBBIN.

CS	A0	RD	WR	REGISTER
0	0	0	1	READ DBBOUT
0	1	0	1	READ STATUS
0	0	1	0	WRITE DBBIN (DATA)
0	1	1	0	WRITE DBBIN (COMMAND)
1	X	X	X	NO ACTION

Figure 4. Register Decoding

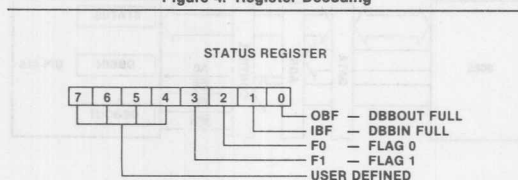


Figure 5. Status Register Format

The remaining four STATUS register bits are user definable. Typical uses of these bits are as status indicators for individual tasks in a multitasking UPI or as UPI generated interrupt status. These bits find a wide variety of uses in the upcoming applications.

Looking into the 8085A from the UPI, the UPI sees the two DBB registers plus the IBF, OBF, and F1 flags. The UPI can write from its accumulator to DBBOUT or read DBBIN into the accumulator. The UPI cannot read OBF, IBF, or F1 directly, but these flags may be tested using conditional jump instructions. The UPI should make sure that OBF is reset before writing new data into DBBOUT to ensure that the master has read previous DBBOUT data. IBF should also be tested before reading DBBIN since DBBIN data is valid only when IBF is set. As was mentioned earlier, the UPI uses F1 to differentiate between command and data contents in DBBIN when IBF is set. The UPI may also write the upper 4-bits of its accumulator to the upper 4-bits of the STATUS register. These bits are thus user definable.

The UPI can test the flags at any time during its internal program execution. It essentially "polls" the STATUS register for changes. If faster response is needed to master commands and data, the UPI's internal interrupt structure can be used. If IBF interrupts are enabled, a master write to DBBIN (either command or data) sets IBF which generates an internal CALL to location 03H in program memory. At this point, working register contents can be saved using bank switching, the accumulator saved in a spare working register, and the DBBIN register read and serviced. The interrupt logic for the IBF interrupt is shown in Figure 6. A few observations concerning this logic are appropriate. Note that if the master writes to DBBIN while the UPI is still servicing the last IBF interrupt (a Return (RETR) instruction has not been executed), the IBF Interrupt Pending line is made high which causes a new CALL to 03H as soon as the first RETR is executed. No ENI (Enable Interrupt) instruction is needed to rearm the interrupt logic as is needed in an 8080 or 8085A system; the RETR performs this function. Also note that executing a DISI to disable further IBF interrupts does not clear a pending interrupt. Only a CALL to location 03H or RESET clears a pending IBF interrupt.

Keeping in mind that the actual master/UPI protocol is dependent on the application, probably the best way to illustrate correct protocol is by example. Let's consider using the UPI as a simple parallel I/O device. (This is a trivial application but it embodies all of the important protocol considerations.) Since the UPI may be either interrupt or non-interrupt driven internally, both cases are considered.

Let's take the easiest configuration first; using the UPI Port 1 as an 8-bit output port. From the UPI's point-of-view, this is an input-only application since all that is required is that the UPI input data from the master. Once the master writes data to the UPI, the UPI reads the DBBIN register and transfers the data to Port 1. No testing for commands vs data is needed since the UPI "knows" it only performs one task — no commands are needed.

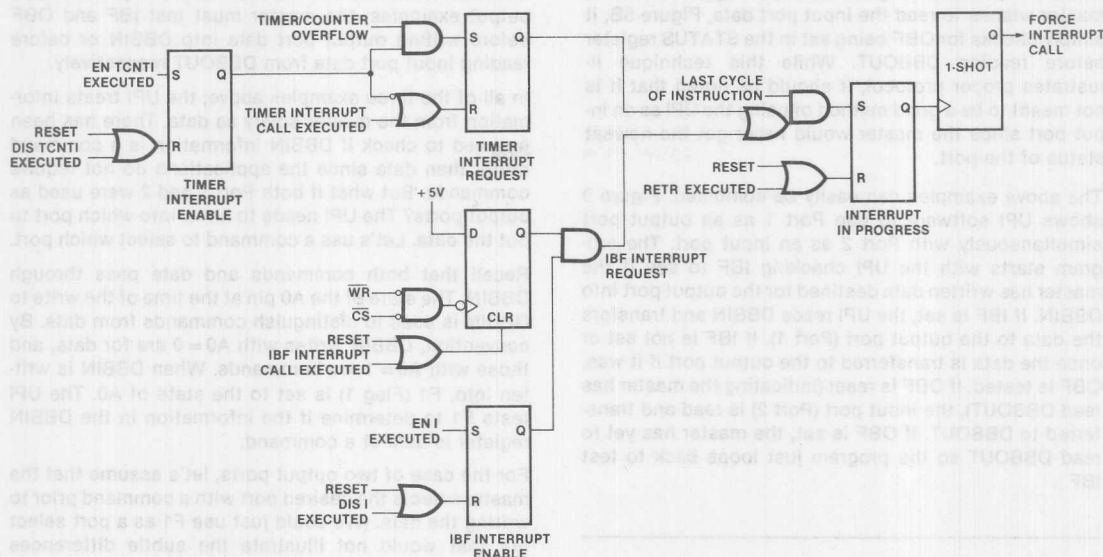


Figure 6. UPI-41A Interrupt Structure

Non-interrupt driven UPI software is shown in Figure 7A while Figure 7B shows interrupt based software. For Figure 7A, the UPI simply waits until it sees IBF go high indicating the master has written a data byte to DBBIN. The UPI then reads DBBIN, transfers it to Port 1, and returns to waiting for the next data. For the interrupt-driven UPI, Figure 7B, once the EN I instruction is executed, the UPI simply waits for the IBF interrupt before handling the data. The UPI could handle other tasks during this waiting time. When the master writes the data to DBBIN, an IBF interrupt is generated which performs a CALL to location 03H. At this point the UPI reads DBBIN (no testing of IBF is needed since an IBF interrupt implies that IBF is set), transfers the data to Port 1, and executes an RETR which returns program flow to the main program.

Software for the master 8085A is included in Figure 7C. The only requirement for the master to output data to the UPI is that it check the UPI to be sure the previous data had been taken before writing new data. To accomplish this the master simply reads the STATUS register looking for IBF = 0 before writing the next data.

```

: UPI INPUT ONLY EXAMPLE - PORT 1 USED AS OUTPUT PORT
: UPI POLLS IBF FOR DATA
:
RESET:  JNIBF  RESET  ; WAIT ON IBF FOR INPUT
        IN    A, DBB  ; INPUT THERE, SO READ IT
        OUTL  P1, A   ; TRANSFER DATA TO PORT 1
        JMP   RESET   ; GO WAIT FOR NEXT DATA

```

Figure 7A. Single Output Port Example — Polling

```

: UPI INPUT ONLY EXAMPLE - PORT 1 USED AS OUTPUT PORT
: DATA INPUT IS INTERRUPT-DRIVEN ON IBF
:
RESET:  EN    I       ; ENABLE IBF INTERRUPTS
        JMP   RESET + 1 ; LOOP WAITING FOR INPUT
IBFINT: IN    A, DBB  ; READ DATA FROM DBBIN
        OUTL  P1, A   ; TRANSFER DATA TO PORT 1
        RETR                ; RETURN WITH RESTORE

```

Figure 7B. Single Output Port Example — Interrupt

```

: 8085 SOFTWARE FOR UPI INPUT-ONLY EXAMPLE
: DATA FOR OUTPUT IS PASSED IN REG. C
:
UPIOUT: IN    STATUS  ; READ UPI STATUS
        ANI    IBF    ; LOOK AT IBF
        JNZ    UPIOUT ; WAIT FOR IBF = 0
        MOV    A, C    ; GET DATA FROM C
        OUT    DBBIN   ; OUTPUT DATA TO DBBIN
        RET                ; DONE, RETURN

```

Figure 7C. 8085A Code for Single Output Port Example

Figure 8A illustrates the case where UPI Port 2 is used as an 8-bit input port. This configuration is termed UPI output-only as the master does not write (input) to the UPI but simply reads either the STATUS or the DBBOUT registers. In this example only the OBF flag is used. OBF signals the master that the UPI has placed new port data in DBBOUT. The UPI loops testing OBF. When OBF is clear, the master has read the previous data and UPI then reads its input port (Port 2) and places this data in DBBOUT. It then waits on OBF until the master reads

DBBOUT before reading the input port again. When the master wishes to read the input port data, Figure 8B, it simply checks for OBF being set in the STATUS register before reading DBBOUT. While this technique illustrates proper protocol, it should be noted that it is not meant to be a good method of using the UPI as an input port since the master would never get the newest status of the port.

The above examples can easily be combined. Figure 9 shows UPI software to use Port 1 as an output port simultaneously with Port 2 as an input port. The program starts with the UPI checking IBF to see if the master has written data destined for the output port into DBBIN. If IBF is set, the UPI reads DBBIN and transfers the data to the output port (Port 1). If IBF is not set or once the data is transferred to the output port if it was, OBF is tested. If OBF is reset (indicating the master has read DBBOUT), the input port (Port 2) is read and transferred to DBBOUT. If OBF is set, the master has yet to read DBBOUT so the program just loops back to test IBF.

```

: UPI OUTPUT ONLY EXAMPLE - PORT 2 USED AS INPUT PORT
: PORT DATA IS AVAILABLE IN DBBOUT
RESET:  JOBF  RESET  ; LOOP IF OBF = 1 (DATA NOT READ)
        IN    A, P2  ; DBBOUT CLEAR, READ PORT
        OUT   DBB, A  ; TRANSFER PORT DATA TO DBBOUT
        JMP   RESET  ; WAIT FOR MASTER TO READ DATA

```

Figure 8A. Single Input Port Example

```

: 8085 SOFTWARE FOR UPI OUTPUT-ONLY EXAMPLE
: INPUT DATA RETURNED IN REG. A
UPIIN:  IN    STATUS  ; READ UPI STATUS
        ANI   OBF    ; LOOK AT OBF
        JZ    UPIIN  ; WAIT UNTIL OBF = 1
        IN    DBBOUT  ; READ DBBOUT
        RET         ; RETURN WITH DATA IN A

```

Figure 8B. 8085A Single Input Port Code

```

: UPI INPUT/OUTPUT EXAMPLE - PORT 1 OUTPUT, PORT 2 INPUT
RESET:  JNIBF  OUT1   ; IF IBF = 0, DO OUTPUT
        IN    A, DBB  ; IF IBF = 1, READ DBBIN
        OUTL  P1, A   ; TRANSFER DATA TO PORT 1
OUT1:   JOBF   RESET  ; IF OBF = 1, GO TEST IBF
        IN    A, P2   ; IF OBF = 0, READ PORT 2
        OUT   DBB, A  ; TRANSFER PORT DATA TO DBBOUT
        JMP   RESET  ; GO CHECK FOR INPUT

```

Figure 9. Combination Output/Input Port Example

The master software is identical to the separate input/output examples; the master must test IBF and OBF before writing output port data into DBBIN or before reading input port data from DBBOUT respectively.

In all of the three examples above, the UPI treats information from the master solely as data. There has been no need to check if DBBIN information is a command rather than data since the applications do not require commands. But what if both Port 1 and 2 were used as output ports? The UPI needs to know into which port to put the data. Let's use a command to select which port.

Recall that both commands and data pass through DBBIN. The state of the A0 pin at the time of the write to DBBIN is used to distinguish commands from data. By convention, DBBIN writes with A0=0 are for data, and those with A0=1 are commands. When DBBIN is written into, F1 (Flag 1) is set to the state of A0. The UPI tests F1 to determine if the information in the DBBIN register is data or a command.

For the case of two output ports, let's assume that the master selects the desired port with a command prior to writing the data. (We could just use F1 as a port select but that would not illustrate the subtle differences between commands and data.) Let's define the port select commands such that bit 1 = 1 if the next data is for Port 1 (Write Port 1 = 0000 0010) and bit 2 = 1 if the next data is for Port 2 (Write Port 2 = 0000 0100). (The number of the set bit selects the port.) Any other bits are ignored. This assignment is completely arbitrary; we could use any command structure, but this one has the advantage of being simple.

Note that the UPI must "remember" from DBBIN write to write which port has been selected. Let's use F0 (Flag 0) for this purpose. If a Write Port 1 command is received, F0 is reset. If the command is Write Port 2, F0 is set. When the UPI finds data in DBBIN, F0 is interrogated and the data is loaded into the previously selected port. The UPI software is shown in Figure 10A.

```

: UPI DUAL OUTPUT PORT EXAMPLE - BOTH PORT 1 AND 2 OUTPUTS
: COMMAND SELECTS DESIRED PORT
: WRITE PORT 1 - 0000 0010 (02H)
: WRITE PORT 2 - 0000 0100 (04H)

```

```

: FLAG 0 USED TO REMEMBER WHICH PORT WAS SELECTED
: BY LAST COMMAND.

```

```

RESET:  JNIBF  RESET  ; WAIT FOR MASTER INPUT
        IN    A, DBB  ; READ INPUT
        JF1   CMD     ; IF F1 = 1, COMMAND INPUT
        JF0   PORT2   ; INPUT IS DATA, TEST F0
        OUTL  P1, A   ; F0 = 0, SO OUTPUT TO PORT 1
        JMP   RESET  ; WAIT FOR NEXT INPUT
PORT2:  OUTL  P2, A   ; F0 = 1, SO OUTPUT TO PORT 2
        JMP   RESET  ; WAIT FOR NEXT INPUT
CMD:    JB1    PT1    ; TEST COMMAND BITS (BIT 1)
        JB2    PT2    ; TEST BIT 2
        JMP   RESET  ; NEITHER BIT SET, WAIT FOR INPUT
PT1:    CLR    F0     ; PORT 1 SELECTED, CLEAR F0
        JMP   RESET  ; WAIT FOR INPUT
PT2:    CLR    F0     ; PORT 2 SELECTED, SET F0
        CPL    F0
        JMP   RESET  ; WAIT FOR INPUT

```

Figure 10A. Dual Output Port Example

Initially, the UPI simply waits until IBF is set indicating the master has written into DBBIN. Once IBF is set, DBBIN is read and F1 is tested for a command. If F1 = 1, the DBBIN byte is a command. Assuming a command, bit 1 is tested to see if the command selected port 1. If so, F0 is cleared and the program returns to wait for the data. If bit 1 = 0, bit 2 is tested. If bit 2 is set, Port 2 is selected so F0 is set. The program then loops back waiting for the next master input. This input is the desired port data. If bit 2 was not set, F0 is not changed and no action is taken.

When IBF = 1 is again detected, the input is again tested for command or data. Since it is necessarily data, DBBIN is read and F0 is tested to determine which port was previously selected. The data is then output to that port, following which the program waits for the next input. Note that since F0 still selects the previous port, the next input could be more data for that port. The port selection command could be thought of as a port select flip-flop control; once a selection is made, data may be repeatedly written to that port until the other port is selected. Master software, Figure 10B, simply must check IBF before writing either a command or data to DBBIN. Otherwise, the master software is straightforward.

For the sake of completeness, UPI software for implementing two input ports is given in Figure 11. This case is simpler than the dual output case since the UPI can assume that all writes to DBBIN are port selection commands so no command/data testing is required. Once the Port Read command is input, the selected port is read and the port data is placed in DBBOUT. Note that in this case F0 is used as a UPI error indicator. If the master happened to issue an invalid command (a command without either bit 1 or 2 set), F0 is set to notify the master that the UPI did not know how to interpret the command. F0 is also set if the master commanded a port read before it had read DBBOUT from the previous command. The UPI simply tests OBF just prior to loading DBBOUT and if OBF = 1, F0 is set to indicate the error.

All of the above examples are, in themselves, rather trivial applications of the UPI although they could easily be incorporated as one of several tasks in a UPI handling multiple small tasks. We have covered them primarily to introduce the UPI concept and to illustrate some master/UPI protocol. Before moving on to more realistic UPI applications, let's discuss two UPI features that do not directly relate to the master/UPI protocol but greatly enhance the UPI's data transfer capability.

In addition to the OBF and IBF bits in the STATUS register, these flags can also be made available directly on two port pins. These port pins can then be used as interrupt sources to the master. By executing an EN FLAGS instruction, Port 2 pin 4 reflects the condition of OBF and Port 2 pin 5 reflects the inverted condition of IBF ($\overline{\text{IBF}}$). These dedicated outputs can then be enabled or disabled via their respective port bit values; i.e., P24 reflects OBF as long as an instruction is executed which sets P24 (i.e. ORL P2,#10H). The same action applies to the $\overline{\text{IBF}}$ output except P25 is used. Thus P24 may serve as a DATA AVAILABLE interrupt output. Like-

wise for P25 as a READY-TO-ACCEPT-DATA interrupt. This greatly simplifies interrupt-driven master-slave data transfers.

```

; 8085 SOFTWARE FOR DUAL OUTPUT PORT EXAMPLE
; THIS ROUTINE WRITES DATA IN REG. C TO PORT 1
; (SAME ROUTINE FOR PORT 2 - JUST CHANGE COMMAND)

PORT1: IN      STATUS      ; READ UPI STATUS
        ANI      IBF        ; LOOK AT IBF
        JNZ      PORT1      ; WAIT UNTIL IBF = 0
        MVI      A, 00000010B ; LOAD WRITE PORT1 CMD
        OUT      UPICMD      ; OUTPUT TO UPI COMMAND PORT
P1:     IN      STATUS      ; READ UPI STATUS AGAIN
        ANI      IBF        ; LOOK AT IBF
        JNZ      P1         ; WAIT UNTIL COMMAND ACCEPTED
        MOV      A, C        ; GET DATA FROM C
        OUT      DBBIN       ; OUTPUT TO DBBIN
        RET                 ; DONE, RETURN

```

Figure 10B. 8085A Dual Output Port Example Code

```

; UPI DUAL INPUT PORT EXAMPLE - BOTH PORT 1 AND 2 INPUTS
; COMMAND SELECTS WHICH PORT IS TO BE READ
; FLAG 0 USED AS ERROR FLAG

RESET:  JNIBF      RESET      ; WAIT FOR INPUT
        CLR      F0          ; CLEAR ERROR FLAG
        IN      A, DBB       ; READ INPUT (COMMAND)
        JB1      PT1         ; TEST BIT 1 (PORT1)
        JB2      PT2         ; TEST BIT 2 (PORT2)
ERROR:   CPL      F0          ; ERROR - COMPLEMENT F0
        JMP      RESET       ; WAIT FOR INPUT
PT1:     IN      A, P1        ; READ PORT 1
        JOBF      ERROR       ; TEST OBF BEFORE LOADING DBBOUT
        OUT      DBB, A       ; LOAD PORT1 DATA INTO DBBOUT
        JMP      RESET       ; WAIT FOR INPUT
PT2:     IN      A, P2        ; READ PORT 2
        JOBF      ERROR       ; TEST OBF BEFORE LOADING DBBOUT
        OUT      DBB, A       ; LOAD PORT2 DATA INTO DBBOUT
        JMP      RESET       ; WAIT FOR INPUT

```

Figure 11. Dual Input Port Example

The UPI also supports a DMA transfer interface. If an EN DMA instruction is executed, Port 2 pin 6 becomes a DMA Request (DRQ) output and P27 becomes a high impedance DMA Acknowledge (DACK) input. Any instruction which would normally set P26 now sets DRQ. DRQ is cleared when DACK is low and either $\overline{\text{RD}}$ or $\overline{\text{WR}}$ is low. When $\overline{\text{DACK}}$ is low, $\overline{\text{CS}}$ and A0 are forced low internally which allows data bus transfers between DBBOUT or DBBIN to occur, depending upon whether $\overline{\text{WR}}$ or $\overline{\text{RD}}$ is true. Of course, the function requires the use of an external DMA controller.

Now that we have discussed the aspects of the UPI protocol and data transfer interfaces, let's move on to the actual applications.

EXAMPLE APPLICATIONS

Each of the following three sections present the hardware and software details of a UPI application. Each application utilizes one of the protocols mentioned in the last section. The first example is a simple 8-digit LED display controller. This application requires only that the UPI perform input operations from the DBBIN; DBBOUT is not used. The reverse is true for the second

application: a sensor matrix controller. The final application involves both DBBOUT and DBBIN operations: a combination serial/parallel I/O device.

The core master processor system with which these applications were developed is the iSBC 80/30 single board computer. This board provides an especially convenient UPI environment since it contains a dedicated socket specifically interfaced for the UPI-41A. The 80/30 uses the 8085A as the master processor. The I/O and peripheral compliment on the 80/30 include 12 vectored priority interrupts (8 on an 8259 Programmable Interrupt Controller and 4 on the 8085A itself), an 8253 Programmable Interval Timer supplying three 16-bit programmable timers (one is dedicated as a programmable baud rate generator), a high speed serial channel provided by a 8251 Programmable USART, and 24 parallel I/O lines implemented with an 8255A Programmable Parallel Interface. The memory compliment contains 16K bytes of RAM using 2117 16K bit Dynamic RAMs and the 8202 Dynamic RAM Controller, and up to 8K bytes of

ROM/EPROM with sockets compatible with 2716, 2758, or 2332 devices. The 80/30's RAM uses a dual port architecture. That is, the memory can be considered a global system resource, accessible from the on-board 8085A as well as from remote CPUs and other devices via the MULTIBUS. The 80/30 contains MULTIBUS control logic which allows up to 16 80/30s or other bus masters to share the same system bus. (More detailed information on the iSBC 80/30 and other iSBC products may be found in the latest Intel Systems Data Catalog.)

A block diagram of the iSBC 80/30 is shown in Figure 12. Details of the UPI interface are shown in Figure 13. This interface decodes the UPI registers in the following format:

Register	Operations
Read STATUS	IN E5H
Write DBBIN (command)	OUT E5H
Read DBBOUT (data)	IN E4H
Write DBBIN (data)	OUT E4H

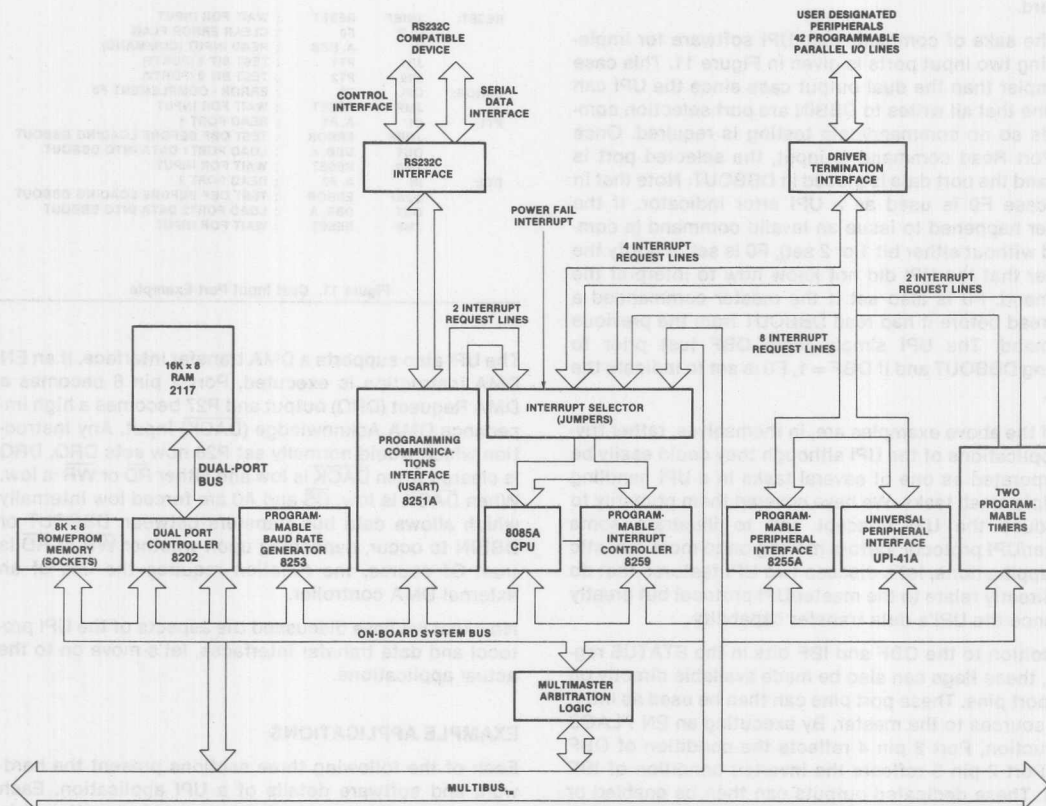


Figure 12. iSBC 80/30 Block Diagram

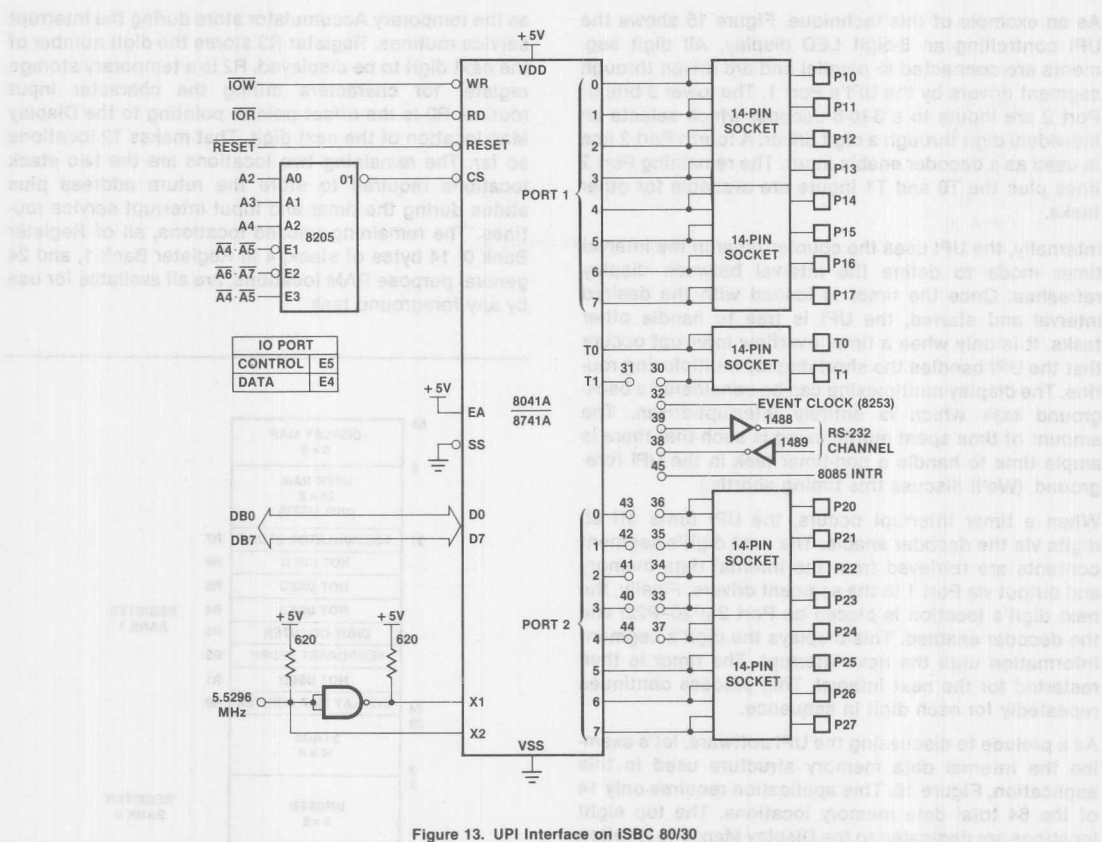


Figure 13. UPI Interface on ISBC 80/30

8-Digit Multiplexed LED Display

The traditional method of interfacing an LED display with a microprocessor is to use a data latch along with a BDC-to-7-segment decoder for each digit of the display. Thus two ICs, seven current limiting resistors, and about 45 connections are required for each digit. These requirements are, of course, multiplied by the total number of digits desired. The obvious disadvantages of this method are high parts count and high power dissipation since each digit is "ON" continuously. Instead, a scheme of time multiplexing the display can be used to decrease both parts count and power dissipation.

Display multiplexing basically involves connecting the same segment (a, b, c, d, e, f, or g) of each digit in parallel and driving the common digit element (anode or cathode) of each digit separately. This is shown schematically in Figure 14. The various digits of the display are not all on at once; rather, only one digit at a time is energized. As each digit is energized, the appropriate segments for that digit are turned on. Each digit is enabled in this way, in sequence, at a rate fast enough to ensure that each digit appears to be "ON" continuously. This implies that the display must be "refreshed" at periodic intervals to keep the digits flicker-free. If the CPU had to handle this task, it would have to suspend normal processing, go update the display, and then return to its nor-

mal flow. This extra burden is ideally handled by a UPI. The master CPU could simply give characters to the UPI and let the UPI do the actual segment decoding, display multiplexing, and refreshing.

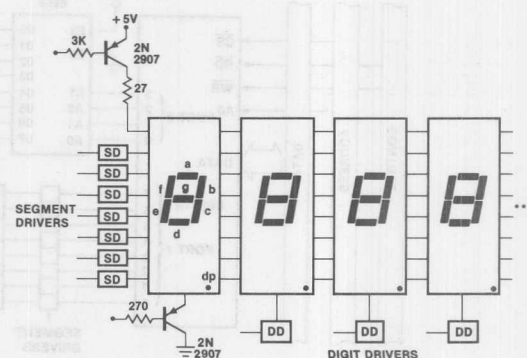


Figure 14. LED Multiplexing

As an example of this technique, Figure 15 shows the UPI controlling an 8-digit LED display. All digit segments are connected in parallel and are driven through segment drivers by the UPI's Port 1. The lower 3 bits of Port 2 are inputs to a 3-to-8 decoder which selects an individual digit through a digit driver. A fourth Port 2 line is used as a decoder enable input. The remaining Port 2 lines plus the T0 and T1 inputs are available for other tasks.

Internally, the UPI uses the counter/timer in the interval timer mode to define the interval between display refreshes. Once the timer is loaded with the desired interval and started, the UPI is free to handle other tasks. It is only when a timer overflow interrupt occurs that the UPI handles the short display multiplexing routine. The display multiplexing can be considered a background task which is entirely interrupt-driven. The amount of time spent multiplexing is such that there is ample time to handle a non-timer task in the UPI foreground. (We'll discuss this timing shortly.)

When a timer interrupt occurs, the UPI turns off all digits via the decoder enable. The next digit's segment contents are retrieved from the internal data memory and output via Port 1 to the segment drivers. Finally, the next digit's location is placed on Port 2 (P20-P22) and the decoder enabled. This displays the digit's segment information until the next interrupt. The timer is then restarted for the next interval. This process continues repeatedly for each digit in sequence.

As a prelude to discussing the UPI software, let's examine the internal data memory structure used in this application, Figure 16. This application requires only 14 of the 64 total data memory locations. The top eight locations are dedicated to the Display Map; one location for each digit. These locations contain the segment and decimal point information for each character. Just how characters are loaded into this section of memory is covered shortly. Register R7 of Register Bank 1 is used

as the temporary Accumulator store during the interrupt service routines. Register R3 stores the digit number of the next digit to be displayed. R2 is a temporary storage register for characters during the character input routine. R0 is the offset pointer pointing to the Display Map location of the next digit. That makes 12 locations so far. The remaining two locations are the two stack locations required to store the return address plus status during the timer and input interrupt service routines. The remaining unused locations, all of Register Bank 0, 14 bytes of stack, 4 in Register Bank 1, and 24 general purpose RAM locations, are all available for use by any foreground task.

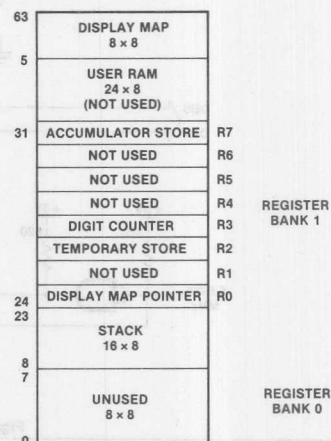


Figure 16. LED Display Controller Data Memory Allocation

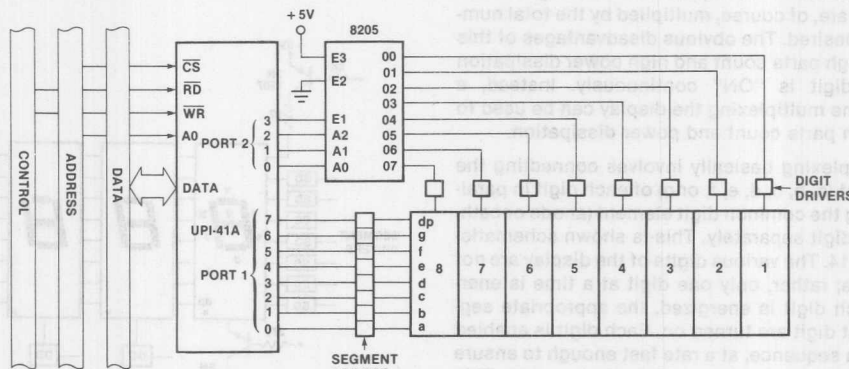


Figure 15. UPI Controlled 8-Digit LED Display

The UPI software consists of only three short routines. One, INIT, is used strictly during initialization. DISPLA is the multiplexing routine called at a timer interrupt. INPUT is the character input handler called at an IBF interrupt. The flow charts for these routines are shown in Figures 17A thru 17C.

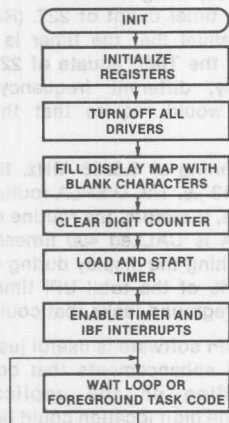


Figure 17A. INIT Routine Flow

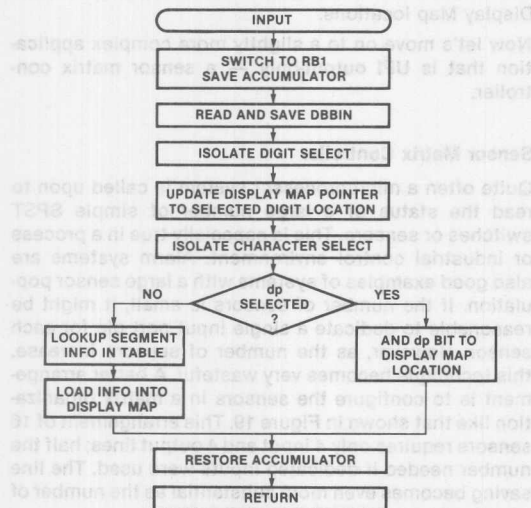


Figure 17B. INPUT Routine Flow

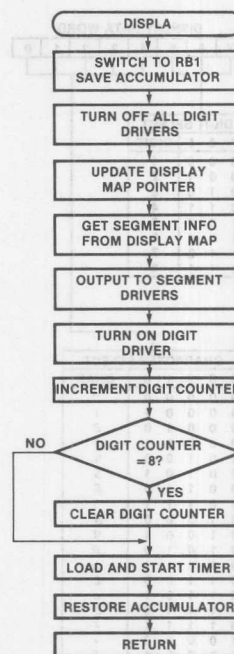


Figure 17C. DISPLA Routine Flow

INIT initializes the UPI by simply turning off all segment and digit drivers, filling the Display Map with blank characters, loading and starting the timer, and enabling both timer and IBF interrupts. Although the flow chart shows the program looping at this point, it is here that the code for any foreground task is inserted. The only restrictions on this foreground task are that it not use I/O lines dedicated to the display and that it not require dedicated use of the timer. It could share the timer if precautions are taken to ensure that the display will still be refreshed at the required interval.

The INPUT routine handles the character input. It is called when an IBF interrupt occurs. After the usual swapping of register banks and saving of the accumulator, DBBIN is read and stored in register R2. DBBIN contains the Display Data Word. The format for this word, Figure 18, has two fields: Digit Select and Character Select. The Digit Select field selects the digit number into which the character from the Character Select field is placed. Notice that the character set is not limited strictly to numerics, some alphanumeric capability is provided. Once DBBIN is read, the offset for the selected digit is computed and placed in the Display Map Pointer R0. Next the segment information for the selected character is found through a look-up table starting in page 3 of the program memory. This segment information is then stored at the location pointed at by the Display Map Pointer. If the Character Select field specified a decimal point, the segment corresponding the decimal point is ANDed into the present segment information for that digit. After the accumulator is restored, execution is returned to the main program.

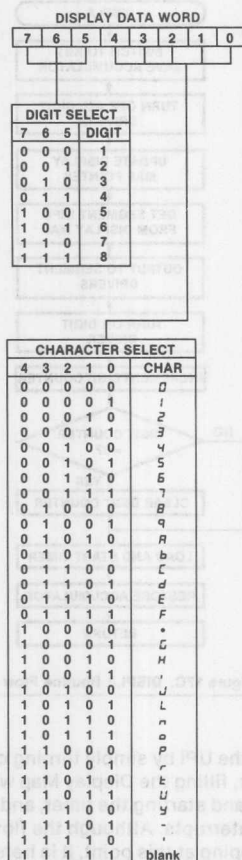


Figure 18. LED Display Controller Display Data Word Format

The DISPLA routine simply implements the multiplexing actions described earlier. It is called whenever a timer interrupt occurs. After saving pre-interrupt status by switching register banks and storing the Accumulator, all digit drivers are turned off. The Display Map Pointer is then updated using the Current Digit Register to point at that digit's segment information in the Display Map. This information is output to Port 1; the segment drivers. The number of the current digit, R3, is then sent to the digit select decoder and the decoder is enabled. This turns on the current digit. The digit counter is incremented and tested to see if all eight digits have been refreshed. If so, the digit counter is reset to zero. If not, nothing is done. Finally, the timer is loaded and restarted, the Accumulator is restored, and the routine returns execution to the main program. Thus DISPLA refreshes one digit each time it is CALLED by the timer interrupt. The digit remains on until the next time DISPLA is executed.

The UPI software listing is included as Appendix A1. Appendix A2 shows the 8085A test routine used to display the contents of a display buffer on the display. The

8085A software takes care of the display digit numbering. Since the application is input-only for the UPI, the only protocol required is that the master must test IBF before writing a Display Data Word into DBBIN.

On the iSBC 80/30, the UPI frequency is at 5.5296 MHz. To obtain a flicker-free display, the whole display must be refreshed at a rate of 50 Hz or greater. If we assume a 50 Hz refresh rate and an 8-digit display, this means the DISPLA routine must be CALLED 50×8 or 400 times/sec. This translates, using the timer interval of $87 \mu s$ at 5.5296 MHz, to a timer count of 227. (Recall from the UPI-41 User's Manual that the timer is an "8-bit up-counter".) Hence the TIME equate of 227D in the UPI listing. Obviously, different frequency sources or display lengths would require that this equate be modified.

With the UPI running at 5.5296 MHz, the instruction cycle time is $2.713 \mu s$. The DISPLA routine requires 28 instruction cycles, therefore, the routine executes in $76 \mu s$. Since DISPLA is CALLED 400 times/sec, the total time spent refreshing the display during one second is then 30 ms or 3% of the total UPI time. This leaves 97.0% for any foreground tasks that could be added.

While the basic UPI software is useful just as it stands, there are several enhancements that could be incorporated depending on the application. Auto-incrementing of the digit location could be added to the input routine to alleviate the need for the master to keep track of digit numbers. This could be (optionally) either right-handed or left-handed entry a la TI or HP calculators. The character set could be easily modified by simply changing the lookup table. The display could be expanded to 16 digits at the expense of one additional Port 2 digit select line, the replacement of the 3-to-8 decoder with a 4-to-16 decoder, and 8 more Display Map locations.

Now let's move on to a slightly more complex application that is UPI output-only — a sensor matrix controller.

Sensor Matrix Controller

Quite often a microprocessor system is called upon to read the status of a large number of simple SPST switches or sensors. This is especially true in a process or industrial control environment. Alarm systems are also good examples of systems with a large sensor population. If the number of sensors is small, it might be reasonable to dedicate a single input port pin for each sensor. However, as the number of sensors increase, this technique becomes very wasteful. A better arrangement is to configure the sensors in a matrix organization like that shown in Figure 19. This arrangement of 16 sensors requires only 4 input and 4 output lines; half the number needed if dedicated inputs were used. The line saving becomes even more substantial as the number of sensors increases.

In Figure 19, the basic operation of the matrix involves scanning individual row select lines in sequence while reading the column return lines. The state of any particular sensor can then be determined by decoding the row and column information. The typical configuration

pulls up the column return lines and the selected row is held low. Deselected rows are held high. Thus a return line remains high for an open sensor on the selected row and is pulled low for a closed sensor. Diode isolation is used to prevent a phantom closure which would occur when a sensor is closed on a selected row and there are two or more closures on a deselected row. Germanium diodes are used to provide greater noise margin at the return line input.

If the main processor was required to control such a matrix it would periodically have to output at the row port and then read the column return port. The processor would need to maintain in memory a map of the previous state of the matrix. A comparison of the new return information to the old information would then be made to determine whether a sensor change had occurred. Any changes would be processed as needed. A row counter and matrix map pointer also require maintenance each scan. Since in most applications sensors change very slowly compared to most processing actions, the processor probably would scan the rows only periodically with other tasks being processed between scans.

Rather than require the processor to handle the rather mundane tasks of scanning, comparing, and decoding the matrix, why not use a dedicated processor? The UPI is perfect.

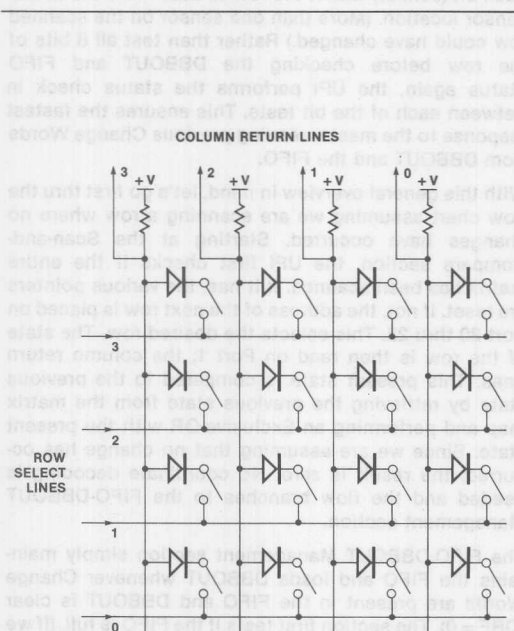


Figure 19. 4 x 4 Sensor Matrix

Figure 20 shows a UPI configuration for controlling up to 128 sensors arranged in a 16 x 8 matrix. The 4-to-16 line decoder is used as the row selector to save port pins and provides the expansion to 128 sensors over the maximum of 64 sensors if the port had been used directly. It also helps increase the port drive capability. The column return lines go directly into Port 1. Features of this design include complete matrix management. As the UPI scans the matrix it compares its present status to the previous scan. If any change is detected, the location of the change is decoded and loaded, along with the sensor's present state, into DBBOUT. This byte is called a Change Word. The Master processor has only to read one byte to determine the status and coordinate of a changed sensor. If the master had not read a previous Change Word in DBBOUT (OBF = 1) before a new sensor change is detected, the new Change Word is loaded into an internal FIFO. This FIFO buffers up to 40 changes before it fills. The status of the FIFO and OBF is made available to the master either by polling the UPI STATUS register, Figure 21A, or as interrupt sources on port pins P24 and P25 respectively, Figure 20. The FIFO NOT EMPTY pin and bit are true as long as there are changes not yet read in the FIFO. As long as the FIFO is not empty, the UPI monitors OBF and loads new Change Words from the FIFO into DBBOUT. Thus, the UPI provides complete FIFO management.

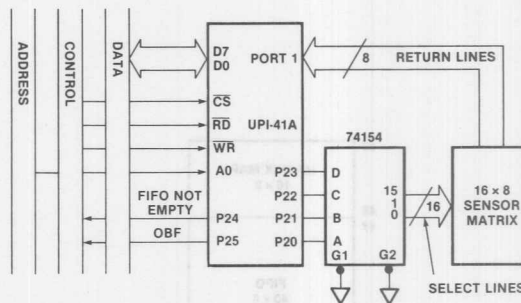


Figure 20. 128 Sensor Matrix Controller

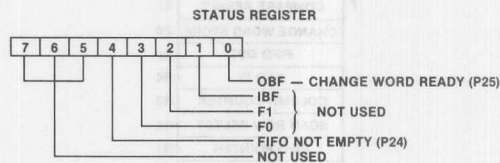


Figure 21A. Sensor Matrix Status Register Format

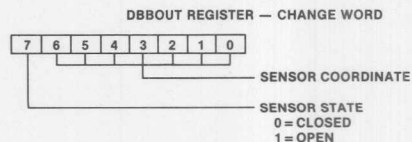


Figure 21B. Sensor Matrix Change Word Format

Internally, the matrix scanning software is programmed to run as a foreground task. This allows the timer/counter to be used by any background task although the hardware configuration leaves only 2 inputs (T0 and T1) plus 2 I/O port pins available. Also, to add a background task, the FIFO would have to be made smaller to accommodate the needed register and data memory space. (It would be possible however to turn the table here and make the scanning software timer/counter interrupt-driven where the timer times the scan interval.)

The data memory organization for this application is shown in Figure 22. The upper 16 bytes form the Matrix Map and store the sensor states from the previous scan; one bit for each sensor. The Change Word FIFO occupies the next 40 locations. (The top and bottom addresses of this FIFO are treated as equate variables in the program so that the FIFO size may easily be changed to accommodate the register needs of other tasks.) Register R0 serves as a pointer into the matrix map area for comparisons and updates of the sensor status. R1 is a general FIFO pointer. The FIFO is implemented as a circular buffer with In and Out pointer registers which are stored in R4 and R5 respectively. These registers are moved into FIFO pointer R1 for actual transfers into or out of the FIFO. R2 is the Row Select Counter. It stores the number of the row being scanned.

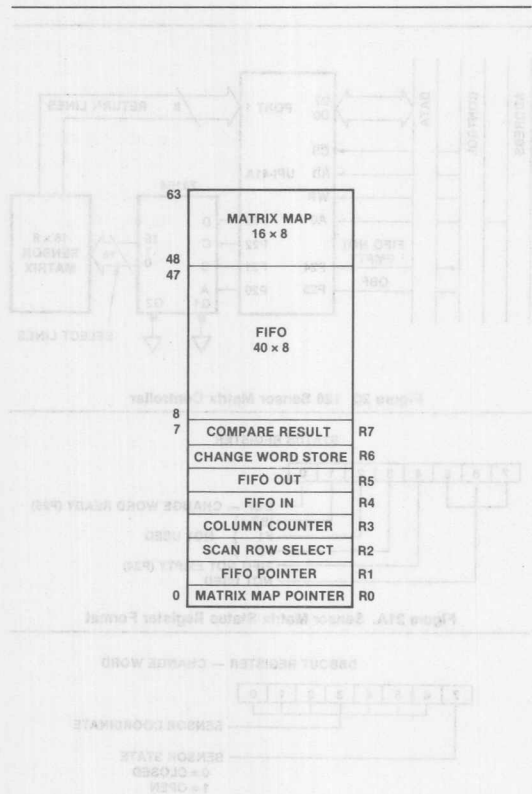


Figure 22. Sensor Matrix Data Memory Map

Register R3 is the Column Counter. This counter is normally set to 00H; however, when a change is detected somewhere in a particular row, it is used to inspect each sensor status bit individually for a change. When a changed sensor bit is found, the Row Select Counter and Column Counter are combined to give the sensor's matrix coordinate. This coordinate is temporarily stored in the Change Word Store, register R6. Register R7 is the Compare Result. As each row is scanned, the return information is Exclusive-OR'd with the return information from the previous scan of that row. The result of this operation is stored in R7. If R7 is zero, there have been no changes on that row. A non-zero result indicates at least one changed sensor.

The basic program operation is shown in the flow chart of Figure 23. At RESET, the software initializes the working registers, the ports, and clears the STATUS register. To get a starting point from which to perform the sensor comparisons, the current status of the matrix is read and stored in the Matrix Map. At this point, the UPI begins looking for changed sensors starting with the first row.

Before delving further into the flow, let's pause to describe the general format of the operation. The UPI scans the matrix one row at a time. If no changes are detected on a particular row, the UPI simply moves to the next row after checking the status of DBBOUT and the FIFO. If a change is detected, the UPI must check each bit (sensor) within the row to determine the actual sensor location. (More than one sensor on the scanned row could have changed.) Rather than test all 8 bits of the row before checking the DBBOUT and FIFO status again, the UPI performs the status check in between each of the bit tests. This ensures the fastest response to the master reading previous Change Words from DBBOUT and the FIFO.

With this general overview in mind, let's go first thru the flow chart assuming we are scanning a row where no changes have occurred. Starting at the Scan-and-Compare section, the UPI first checks if the entire matrix has been scanned. If it has, the various pointers are reset. If not, the address of the next row is placed on Port 20 thru 23. This selects the desired row. The state of the row is then read on Port 1; the column return lines. This present state is compared to the previous state by retrieving the previous state from the matrix map and performing an Exclusive-OR with the present state. Since we are assuming that no change has occurred, the result is zero. No coordinate decoding is needed and the flow branches to the FIFO-DBBOUT Management section.

The FIFO-DBBOUT Management section simply maintains the FIFO and loads DBBOUT whenever Change Words are present in the FIFO and DBBOUT is clear (OBF = 0). The section first tests if the FIFO is full. (If we assume our "no-change" row is the first row scanned, the FIFO obviously would not be full.) If it is, the UPI waits until OBF=0, at which point the next Change Word is retrieved from the FIFO and placed in DBBOUT. This "unfills" the FIFO making room for more Change Words. At this point, the Column Counter, R3, is checked. For rows with no changes, the Column

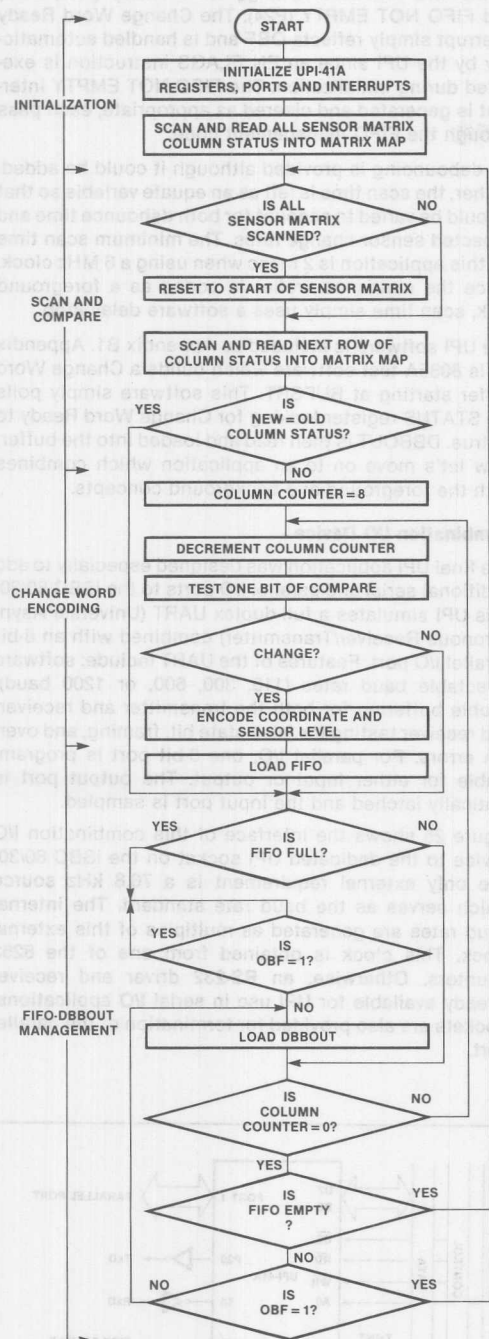


Figure 23. Sensor Matrix Controller Flow Chart

Counter is always zero so the test simply falls through. (We cover the case for changes shortly.) Now the FIFO is tested for being empty. If it is, there is no sense in any further tests so the flow simply goes back up to scan the next row. If the FIFO is not empty, DBBOUT is tested again through OBF. If a Change Word is in DBBOUT waiting for the master to read it, nothing can be done and the flow likewise branches up for the next row. However, if the DBBOUT is free and remembering that the previous test showed that the FIFO was not empty, DBBOUT is loaded with the next Change Word and the last two conditional tests repeat.

Now let's assume the next row contains several changed sensors. Like before, the row is selected, the return lines read, and the sensor status compared to the previous scan. Since changes have occurred, the Exclusive-OR result is now non-zero. Any 1s in the result reflect the positions of the changed sensors. This non-zero result is stored in the Compare Result register, R7. At this point, the Column Counter is preset to 8. To determine the changed sensors' locations, the Compare Result register is shifted bit-by-bit to the left while decrementing the Column Counter. After each shift, bit 7 of the result is tested. If it is a one, a changed sensor has been found. The Column Counter then reflected the sensor's matrix column position while the Scan Row Select register holds its row position. These registers are then combined in R6, the Change Word Store, to form the sensor's matrix coordinate section of the Change Word. The 8th bit of the Change Word Store is coded with the sensor's present state (Figure 21). This byte forms the complete Change Word. It is loaded into the next available FIFO position. If bit 7 of the Compare Result had been a zero, that particular sensor had not changed and the coordinate decoding is not performed.

In between each shift, test, and coordinate encode (if necessary), the FIFO-DBBOUT Management is performed. It is the Column Counter test within this section that routes the flow back up to the Change Word Encoding section if the entire Compare Result (row) has not been shifted and tested.

The FIFO is implemented as a circular buffer with IN and OUT pointers (R4 and R5 respectively). The operations of the FIFO is best understood using an example, Figure 24. This series of figures show how the FIFO, DBBOUT, and OBF interact as changes are detected and Change Words are read by the master. The letters correspond to sequential Change Words being loaded into the FIFO. Note that the figures show only a 4 x 8 FIFO however, the principles are the same in the 40 x 8 FIFO.

Figure 24A shows the condition where no Change Words have been loaded into the FIFO or DBBOUT. In Figure 24B a change, "A", has been detected, decoded, and loaded into the FIFO at the location equal to the value of the FIFO-IN pointer. The FIFO-IN pointer is then incremented and the FIFO-OUT pointer is reset to the bottom of the FIFO since it had reached the FIFO top. Now that a Change Word is in the FIFO, OBF is checked to see if DBBOUT is empty. Because OBF = 0, DBBOUT is empty and the Change Word is loaded from the FIFO location pointed at by the FIFO-OUT pointer. This is shown in Figure 24C. Loading DBBOUT automatically

Change Word is finally read by the master resetting OBF. This allows the next Change Word to be loaded into DBBOUT. Note that each time the FIFO is loaded, the FIFO-IN pointer increments. Each time DBBOUT is read the FIFO-OUT pointer increments unless there are no more Change Words in the FIFO. Both pointers wrap-around to the bottom once they reach the FIFO top. The remaining figures show more Change Words being loaded into the FIFO. When the entire FIFO fills and DBBOUT can not be loaded (OBF = 1), scanning stops until the master reads DBBOUT making room for more Change Words.

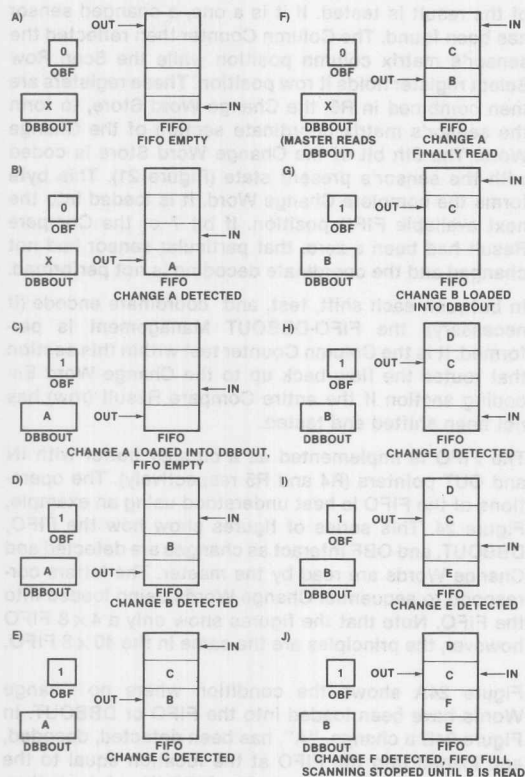


Figure 24A-J. FIFO Operation Example

interrupt simply reflects OBF and is handled automatically by the UPI since an EN FLAGS instruction is executed during initialization. The FIFO NOT EMPTY interrupt is generated and cleared as appropriate, each pass through the FIFO management code.

No debouncing is provided although it could be added. Rather, the scan time is left as an equate variable so that it could be varied to account for both debounce time and expected sensor change rates. The minimum scan time for this application is 2 msec when using a 6 MHz clock. Since the matrix controller is coded as a foreground task, scan time simply uses a software delay loop.

The UPI software is included as Appendix B1. Appendix B2 is 8085A test software which builds a Change Word buffer starting at BUFSRT. This software simply polls the STATUS register looking for Change Word Ready to go true. DBBOUT is then read and loaded into the buffer. Now let's move on to an application which combines both the foreground and background concepts.

Combination I/O Device

The final UPI application was designed especially to add additional serial and parallel I/O ports to the iSBC 80/30. This UPI simulates a full-duplex UART (Universal Asynchronous Receiver/Transmitter) combined with an 8-bit parallel I/O port. Features of the UART include: software selectable baud rates (110, 300, 600, or 1200 baud), double buffering for both the transmitter and receiver, and receiver testing for false state bit, framing, and overrun errors. For parallel I/O, one 8-bit port is programmable for either input or output. The output port is statically latched and the input port is sampled.

Figure 25 shows the interface of this combination I/O device to the dedicated UPI socket on the iSBC 80/30. The only external requirement is a 76.8 kHz source which serves as the baud rate standard. The internal baud rates are generated as multiples of this external clock. This clock is obtained from one of the 8253 counters. Otherwise, an RS-232 driver and receiver already available for UPI use in serial I/O applications. Sockets are also provided for termination of the parallel port.

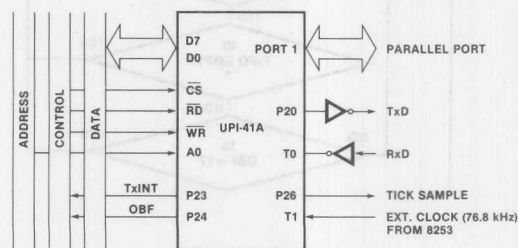


Figure 25. Combination I/O Device

There are three commands for this application. Their format is shown in Figure 26. The CONFIGURE command specifies the serial baud rate and the parallel I/O direction. Normally this command is issued once during system initialization. The I/O command causes a parallel I/O operation to be performed. If the parallel port direction is output, the UPI expects the data byte immediately following an I/O command to be data for the output port. If the port is in the input direction, an I/O command causes the port to be read and the data placed in DBBOUT. The RESET ERROR command resets the serial receiver error bits in the STATUS register.

The STATUS register format is shown in Figure 27. Looking at each bit, bit 0 (OBF) is the DATA AVAILABLE flag. It is set whenever the UPI places data into DBBOUT. Since the data may come from either the receiver or the parallel input port, the F0 and F1 flags (bits 2 and 3) code the source. Thus, when the master finds OBF set, it must decode F0 and F1 to determine the source.

Bit 1 (IBF) functions as a busy bit. When IBF is set, no writes to DBBIN are allowed. Bit 5 is the TxINT (Transmitter Interrupt) bit. It is asserted whenever the transmitter buffer register is empty. The master uses this bit to determine when the transmitter is ready to accept a data character.

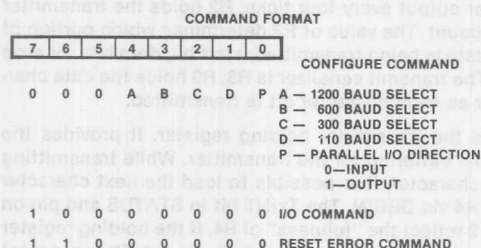


Figure 26. Combination I/O Command Format

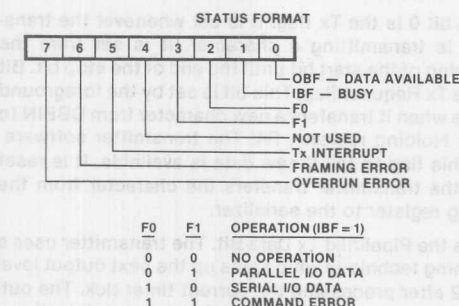


Figure 27. STATUS Register Format

Bits 6 and 7 are receiver error flags. The framing error flag, bit 6, is set whenever a character is received with an invalid stop bit. Bit 7, overrun error, is set if a character is received before the master has read a previous character. If an overrun occurs, the previous character is overwritten and lost. Once an error occurs, the error flag remains set until reset by a RESET ERROR command. A set error flag does not inhibit receiver operation however.

Figure 28 shows the port pin definition for this application. Port 1 is the parallel I/O port. The UART uses Port 2 and the Test inputs. P20 is the transmitter data out pin. It is set for a mark and reset for a space. P23 is a transmitter interrupt output. This pin has the same timing as the TxINT bit in the STATUS register. It is normally used in interrupt-driven systems to interrupt the master processor when the transmitter is ready to accept a new data character.

The OBF flag is brought out on P24 as a master interrupt when data is available in DBBOUT. P26 is a diagnostic pin which pulses at four times the selected baud rate. (More about this pin later.) The receiver data input uses the T0 input. One of the Port 2 pins could have been used, however, the software can test the T0 in one instruction without first reading a port.

The T1 input is the baud rate external source. The UART divides this input to determine the timing needed for the selected baud rate. The input is a non-synchronous 76.8 kHz source.

Internally, when the CONFIGURE command is received and the selected baud rate is determined, the internal timer/counter is loaded with a baud rate constant and started in the event counter mode. Timer/counter interrupts are then enabled. The baud rate constant is selected to provide a counter interrupt at four times the desired baud rate. At each interrupt, both the transmitter and receiver are handled. Between interrupts, any new commands and data are recognized and executed.

PORT PIN DEFINITION		
PORT	BIT	FUNCTION
1	0-7	PARALLEL I/O
	2	Tx DATA
		NOT USED
		NOT USED
		Tx INTERRUPT
	4	OBF INTERRUPT
		NOT USED
		NOT USED (TICK SAMPLE)
T0	7	NOT USED
		Rx DATA
T1		EXTERNAL CLOCK (76.8 kHz)

Figure 28. Combination I/O Port Definition

As a prelude to discussing the flow charts, Figure 29 shows the register definition. Register Bank 0 serves the UART receiver and parallel I/O while Register Bank 1 handles the UART transmitter and commands. Looking at RB0 first, R3 is the receiver status register, RxSTS. Reflected in the bits of this register is the current receiver status in sequential order. Figure 30 shows this bit definition. Bit 0 is the Rx flag. It is set whenever a possible start bit is received. Bit 1 signifies that the start bit is good and character construction should begin with the next received bit. Bit 2 is the Good Start flag. Bit 3 is the Byte Finished flag. When all data bits of a character are received, this flag is set. When all the bits, data and stop bits are received, the assembled character is loaded into the holding register (R4 in Figure 29) bit 3, the Data Ready flag, is set. The foreground routine which looks for commands and data continuously, looks at this bit to determine when the receiver has received a character. Bits 4 and 5 signify any error conditions for a particular character.

The parallel I/O port software uses bits 6 and 7. Bit 6 codes the I/O direction specified by the last CONFIGURE command. Bit 7 is set whenever an I/O command is received. The foreground routine tests this bit to determine when an I/O operation has been requested by the master.

As was mentioned, R4 is the receiver holding register. Assembled characters are held in this register until the foreground routine finds DBBOUT free, at which time the data is transferred from R4 to DBBOUT. R5 is the receiver tick counter. Recall that counter interrupts occur at four times the baud rate. Therefore, once a start bit is found, the receiver only needs to look at the data every four interrupts or tick counts. R5 holds the current tick count.

63	USER RAM (NOT USED)		
32			
31	AC TEMP. STORE	R7	
30	COMMAND STORE	R6	
29	Tx STATUS—TxSTS	R5	
28	Tx BUFFER	R4	REGISTER BANK 1
27	Tx SERIALIZER	R3	
26	Tx TICK COUNTER	R2	
25	BAUD RATE CONSTANT	R1	
24	NOT USED	R0	
23	STACK (ONE LEVEL USED)		
8			
7	STATUS STORE	R7	
6	Rx DESERIALIZER	R6	
5	Rx TICK COUNTER	R5	
4	Rx HOLDING	R4	REGISTER BANK 0
3	Rx STATUS—RxSTS	R3	
2	NOT USED	R2	
1	NOT USED	R1	
0	NOT USED	R0	

Figure 29. Combination I/O Register Map

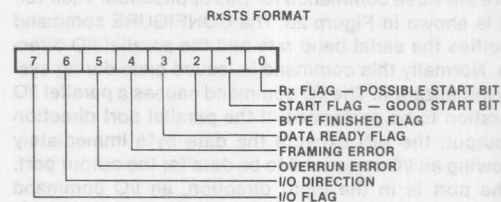


Figure 30. RxSTS Register

R6 is the receiver de-serializing register. Data characters are assembled in this register. R6 is preset to 80H when a good start bit is received. As each bit is sampled every four timer ticks, they are rotated into the leftmost bit of R6. The software knows the character assembly is complete when the original preset bit rotates into the carry.

An image of the upper 4 bits of the STATUS register is stored in R7. These bits are the TxINT, Framing and Overrun bits. This image is needed since the UPI may load the upper 4 STATUS register bits from its accumulator; however, it cannot read STATUS directly.

In Register Bank 1 (Figure 29), R1 holds the baud rate constant which is found from decoding the baud rate select bits of the CONFIGURE command. The counter is reloaded with this constant every timer tick. Like the receiver, the transmitter only needs to update the transmitter output every four ticks. R2 holds the transmitter tick count. The value of R2 determines which portion of the data is being transmitted; start bit, data bits, or stop bit. The transmit serializer is R3. R3 holds the data character as each character bit is transmitted.

R4 is the transmitter holding register. It provides the double buffering for the transmitter. While transmitting one character, it is possible to load the next character into R4 via DBBIN. The TxINT bit in STATUS and pin on Port 2 reflect the "fullness" of R4. If the holding register is empty, the interrupt bit and pin are set. They are reset when the master writes a new data byte for the transmitter into DBBIN. The transmitter Status register (TxSTS) is R5. Like RxSTS, TxSTS contains flag bits which indicate the current state of the transmitter. This flag bit format is shown in Figure 31.

TxSTS bit 0 is the Tx flag. It is set whenever the transmitter is transmitting a character. It is set from the beginning of the start bit until the end of the stop bit. Bit 1 is the Tx Request flag. This bit is set by the foreground routine when it transfers a new character from DBBIN to the Tx Holding register, R4. The transmitter software uses this flag to tell if new data is available. It is reset when the transmitter transfers the character from the holding register to the serializer.

Bit 2 is the Pipelined Tx Data Bit. The transmitter uses a pipelining technique which sets up the next output level in bit 2 after processing the current timer tick. The output level is always changed at the same point after a timer tick interrupt. This technique ensures that no bit timing distortion results from different length processing paths through the receiver and transmitter routines.

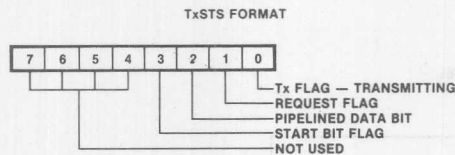


Figure 31. TxSTS Register

Bit 3 of TxSTS is the Start Bit flag. It is set by the transmitter when the start bit space is set up in the Pipelined Data Bit. This allows the transmitter to differentiate between the start bit and data bits on following timer ticks.

The flow charts for this application are shown in Figures 32A-F. At reset, the INIT routine is executed which initializes the registers and port pins. After initialization, IBF and OBF are tested in MNLOOP. These flags are tested continually in this loop. If IBF is set, F1 is tested for command or data and execution is transferred to the appropriate routine (CMD or DATA). If IBF = 0, OBF is checked. If OBF = 0 (DBBOUT is free), the Rx Data Ready and I/O flags in RxSTS are tested. If Rx Data Ready is set, the received data is retrieved from the Rx Holding register and transferred to DBBOUT. Any error flags associated with that data are also transferred to STATUS. If the I/O flag is set and the I/O direction is input, Port 1 is read and the data transferred to DBBOUT. In either case, F0 and F1 are set to indicate the data source.

If IBF is set by a command write to DBBIN, CMD reads the command and decodes the desired operation. If an

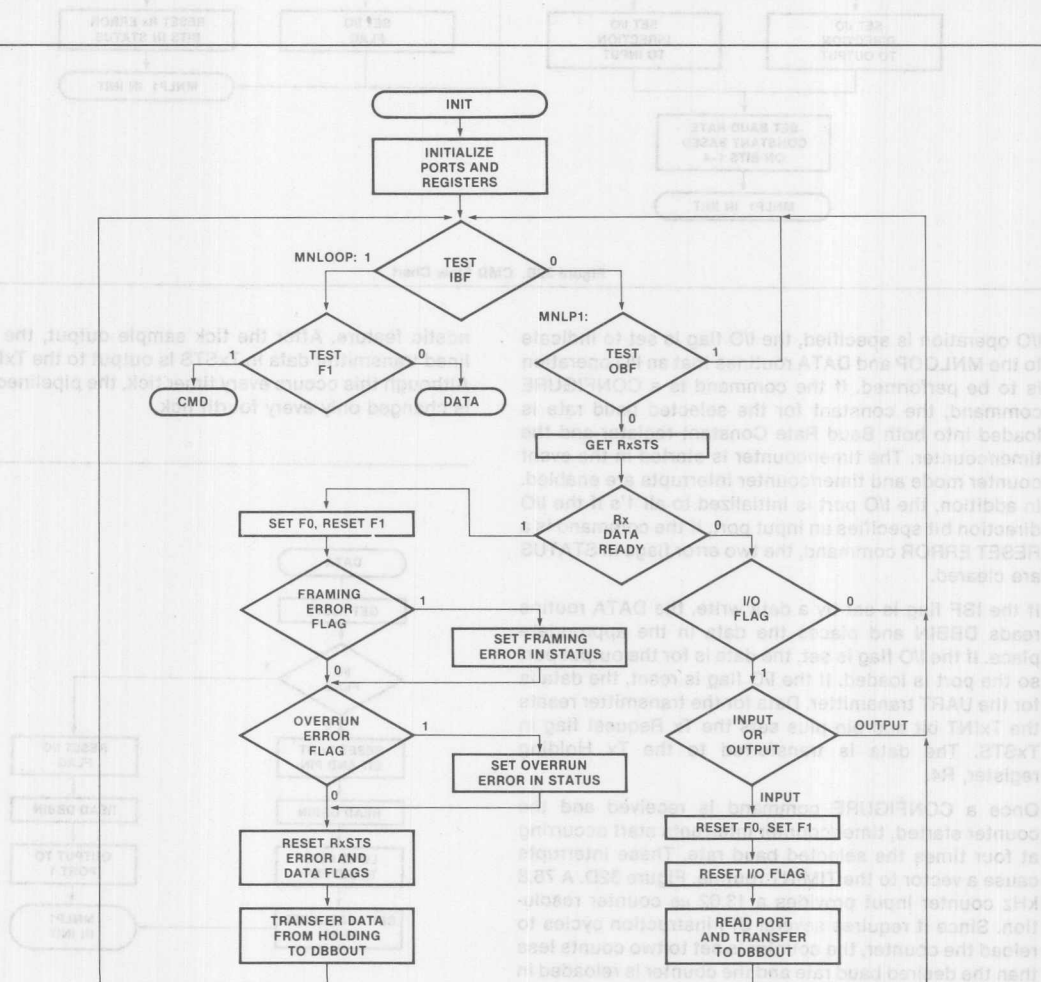


Figure 32A. INIT Flow Chart

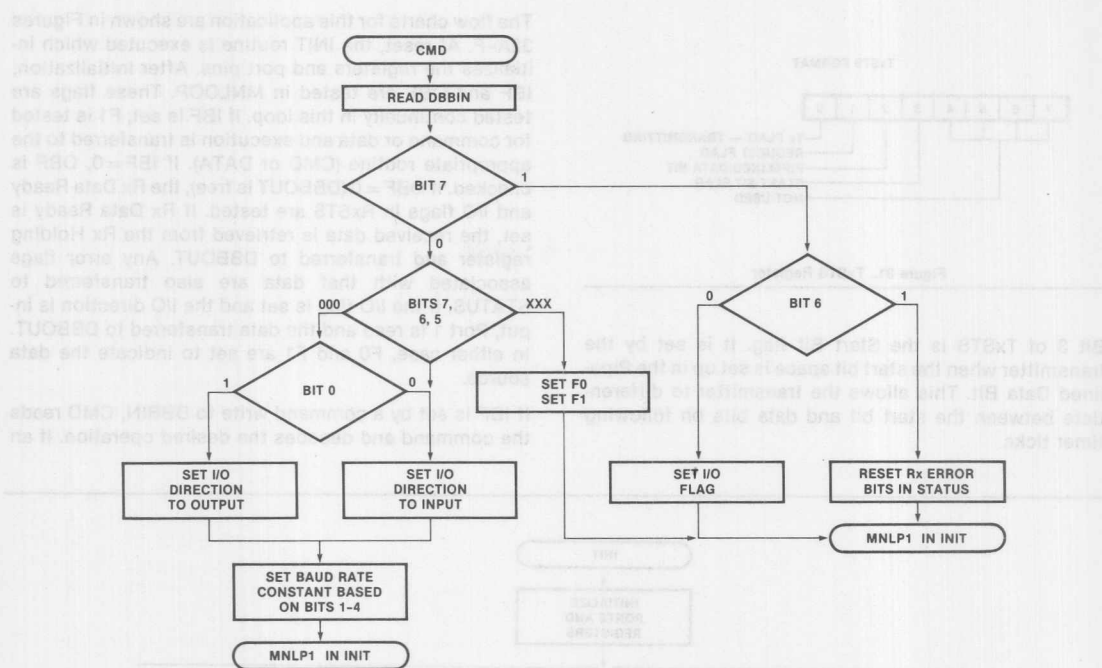


Figure 32B. CMD Flow Chart

I/O operation is specified, the I/O flag is set to indicate to the MNLOOP and DATA routines that an I/O operation is to be performed. If the command is a CONFIGURE command, the constant for the selected baud rate is loaded into both Baud Rate Constant register and the timer/counter. The timer/counter is started in the event counter mode and timer/counter interrupts are enabled. In addition, the I/O port is initialized to all 1's if the I/O direction bit specifies an input port. If the command is a RESET ERROR command, the two error flags in STATUS are cleared.

If the IBF flag is set by a data write, the DATA routine reads DBBIN and places the data in the appropriate place. If the I/O flag is set, the data is for the output port so the port is loaded. If the I/O flag is reset, the data is for the UART transmitter. Data for the transmitter resets the TxINT bit and pin plus sets the Tx Request flag in TxSTS. The data is transferred to the Tx Holding register, R4.

Once a CONFIGURE command is received and the counter started, timer/counter interrupts start occurring at four times the selected baud rate. These interrupts cause a vector to the TIMINT routine, Figure 32D. A 76.8 kHz counter input provides a 13.02 μ s counter resolution. Since it requires several UPI instruction cycles to reload the counter, the counter is set to two counts less than the desired baud rate and the counter is reloaded in TIMINT synchronous with the second low-going transition after the interrupt. Once the counter is reloaded, an output port (P26) is toggled to give an external indication of internal counter interval. This is a helpful diag-

nostic feature. After the tick sample output, the pipelined transmitter data in TxSTS is output to the Tx pin. Although this occurs every timer tick, the pipelined data is changed only every fourth tick.

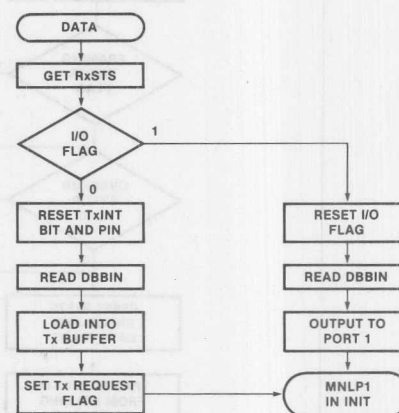


Figure 32C. Data Flow Chart

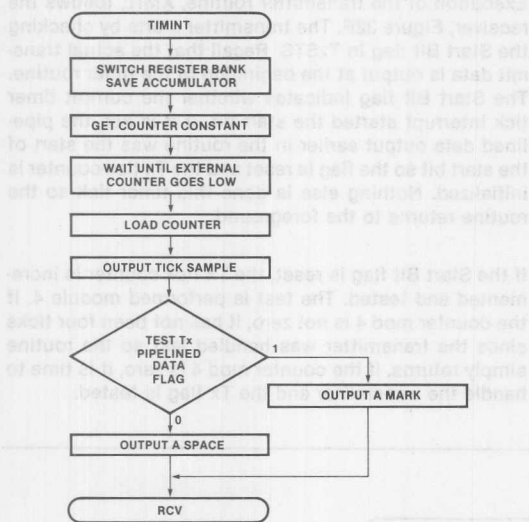


Figure 32D. TIMINT Flow Chart

The receiver is now handled, Figure 32E. The RX flag in RxSTS is examined to see if the receiver is currently in the process of receiving a character. If it is not, the RxD input is tested for a space condition which might indicate a possible start bit. If the input is a mark, no start bit is possible and execution branches to the transmitter flow, XMIT. If the input is a space, the Rx flag is set before proceeding with XMIT.

If the Rx flag is found set when entering RCV, the receiver is in the process of receiving a character. If so, the Start Bit flag is then tested to determine if a good start bit so the Start Bit flag is set, the Rx Tick Counter is initialized to four, and the Rx De-serializer initialized to 80H. A mark indicates a bad start bit so the Rx flag is reset to abort the reception.

start bit so the Start Bit flag is set, the Rx tick counter is initialized to four, and the Rx deserializer initialized to 80H. A mark indicates a bad start bit so the Rx flag is reset to abort the reception.

If the Start Bit flag is set, the program is somewhere in the middle of the received character. Since the data should be sampled every fourth timer tick, the Tick Counter is decremented and tested for zero. If non-zero no sample is needed and execution continues with

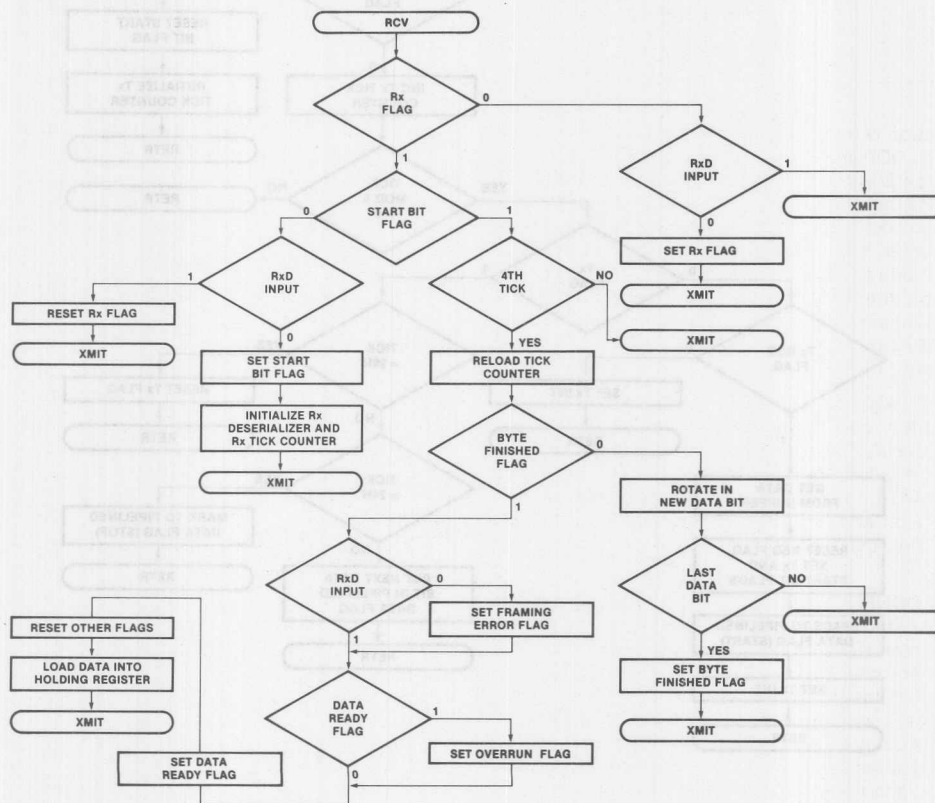


Figure 32E. RCV Flow Chart

XMIT. If zero, the tick counter is reset to four. Now the Byte Finished flag is tested to determine if the data sample is a data or stop bit. If reset, the sample is a data bit. The sample is done and the new bit rotated into the Rx deserializer. If this rotate sets the carry, that data bit was the last so the Byte Finished flag is set. If the carry is reset, the data bit is not the last so execution simply continues with XMIT.

Had the Byte Finished flag been set, this sample is for the stop bit. The Rx D input is tested and if a space, the Framing Error flag is set. Otherwise, it is reset. Next, the Rx Data Ready flag is tested. If it is set, the master has not read the previous character so the Overrun Error flag is set. Then the Rx Data Ready flag is set and the received data character is transferred into the Rx Holding register. The Rx, Start Bit, and Byte Finished flags are reset to get ready for the next character.

Execution of the transmitter routine, XMIT, follows the receiver, Figure 32F. The transmitter starts by checking the Start Bit flag in TxSTS. Recall that the actual transmit data is output at the beginning of the timer routine. The Start Bit flag indicates whether the current timer tick interrupt started the start bit. If it is set, the pipelined data output earlier in the routine was the start of the start bit so the flag is reset and the Tx tick counter is initialized. Nothing else is done this timer tick so the routine returns to the foreground.

If the Start Bit flag is reset, the Tx tick counter is incremented and tested. The test is performed module 4. If the counter mod 4 is not zero, it has not been four ticks since the transmitter was handled last so the routine simply returns. If the counter mod 4 is zero, it is time to handle the transmitter and the Tx flag is tested.

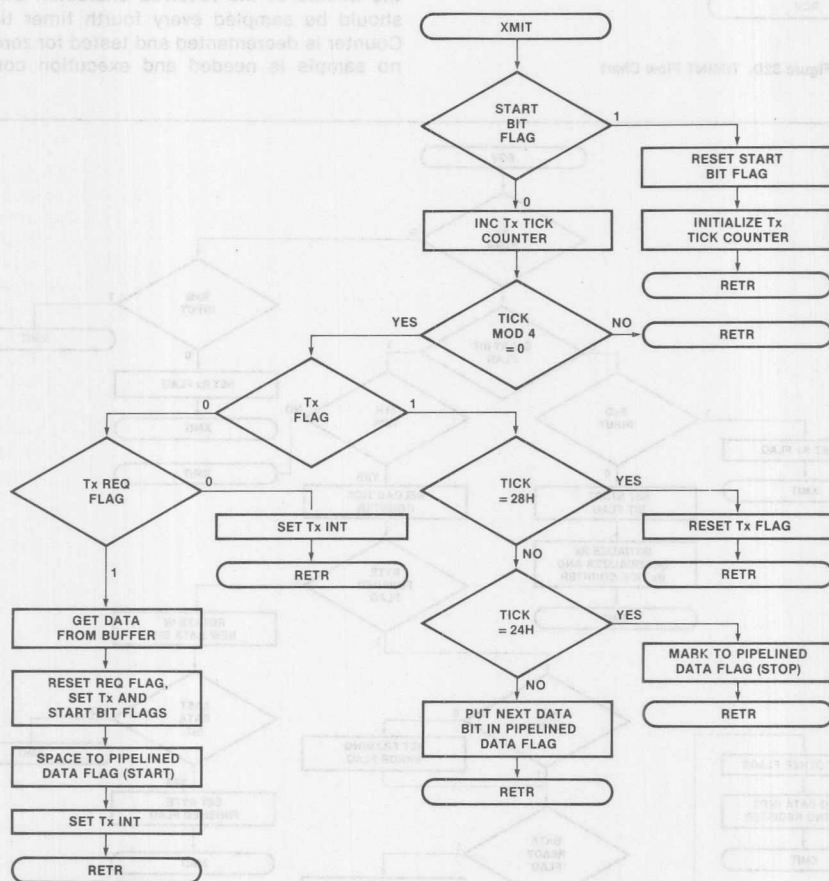


Figure 32F. XMIT Flow Chart

The Tx flag indicates whether the transmitter is active. If the transmitter is inactive, no character is currently being transmitted so the Tx Request flag is tested to see if a new character is waiting in the Tx buffer. If no character is waiting (Tx Request Flag = 0), the Tx interrupt pin and bit are set before returning to the foreground. If there is a character waiting, it is retrieved from the buffer and placed in the Tx serializer. The Tx Request flag is reset while the Tx and Start Bit flags are set. A space is placed in the Tx Pipelined Data bit so a start bit will be output on the next tick. Since the Tx buffer is now empty, the Tx interrupt bit and pin are set to indicate the availability of the buffer to the master. The routine then returns to the foreground.

If the tick counter mod 4 is zero and the Tx flag indicates the transmitter is in the middle of a character, the tick counter is checked to see what transmitter operation is needed. If the counter is 28H (40D), all data bits plus the stop bits are complete. The character is therefore done and the Tx flag is reset. If the counter is 24H (36D), the data bits are complete and the next output should be a mark for the stop bit so a mark is loaded into the Tx Pipelined Data bit.

If neither of the above conditions are met for the counter, the transmitter is some place in the data field, so the next data bit is rotated out of the Tx serializer into the Pipelined Data bit. The next tick outputs this bit.

At this point the program execution is returned to the foreground.

That completes the discussion of the combination I/O device flow charts. The UPI software listing is shown in Appendix C1. Appendix C2 is example 8085A driver software.

Several observations concerning the drivers are appropriate. Notice that since the receiver and input port of the UPI use the OBF flag and interrupt output, the interrupt and flag are cleared when the master reads DBBOUT. This is not true for the transmitter. There is always some time after a master write of new transmitter data before the transmitter interrupt bit and pin are cleared. Thus in an interrupt-driven system, edge-sensitive interrupts should be used. For polled-systems, the software must wait after writing new data for IBF = 0 before re-examining the Tx Interrupt flag in STATUS.

Notice that this application uses none of the user Data Memory above Register Bank 1 and only 361 bytes of Program Memory. This leaves the door open for many improvements. Improvements that come to mind are increased buffering of the transmit or received data, modem control pins, and parallel port handshaking inputs.

This completes our discussion of specific UPI applications. Before concluding, let's look briefly at two debug techniques used during the development of these applications that you might find useful in your own designs.

DEBUG TECHNIQUES

Since the UPI is essentially a single-chip microcomputer, the classical data, address, and control buses are

not available to the outside world during normal operation. This fact normally makes debugging a UPI design difficult; however, certain "tricks" can be included in the UPI software to ease this task.

If a UPI is handling multiple tasks, it is usually easier to code and debug each task individually. This is fairly standard procedure. Since each task usually utilizes only a subset of the total number of I/O pins, coding only one task leaves some I/O pins free. Port output instructions can then be added in the task code being debugged which toggle these unused pins to determine which section of task code is being executed at any particular time. The task can also be made to "wait" at various points by using an extra pin as an input and adding code to loop until a particular input condition is met.

One example of using an extra pin as an output is included in the combination serial/parallel device code. During initial development the receiver was not receiving characters correctly. Since this could be caused by incorrect sampling, three lines of code were added to toggle bit 6 of Port 2 at each tick of the sample clock. This code is at lines 184 and 185 of the listing. Thus by looking at the location of the tick sample pulse with respect to the received bit, the UPI sampling interval can be observed. The tick sample time was incorrect and the code was modified accordingly. Similar techniques could be applied at other locations in the program.

The EPROM version of the UPI (8741A) also contains another feature to aid in debug: the capability to single step thru a program. The user may step thru the program instruction-by-instruction. The address of the next instruction to be fetched is available on Port 1 and the lower 2 bits of Port 2. Figure 33 shows the timing used in the discussion below. When the Single Step input, \overline{SS} , is brought low, the internal processor responds by stopping during the fetch portion of the next instruction. This action is acknowledged by the processor raising the SYNC output. The address of the instruction to be fetched is then placed on the port pins. This state may be held indefinitely. To step to the next instruction, \overline{SS} is raised high, which causes SYNC to go low, which is then used to return \overline{SS} low. This allows the processor to advance to the next instruction. If \overline{SS} is left high, the processor continues to execute at normal speed until \overline{SS} goes low.

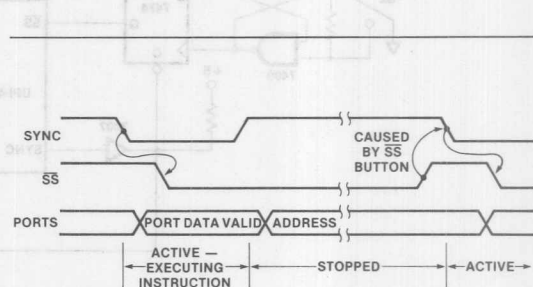


Figure 33. Single Step Timing

When the S2 is depressed, the 7474 is set which causes SS to go high. This allows the processor to fetch and execute the instruction whose address was displayed. SYNC going low during execution, clears the 7474 lowering SS. Thus the processor again stops when execution is complete and the next fetch is started.

Well, that's it. Machine readable (floppy disk or paper

- 8278 Keyboard Display Controller

- Other pre-programmed UPIs are the 8294 Data Encryp-

For information about Lacite, write to:

Insite

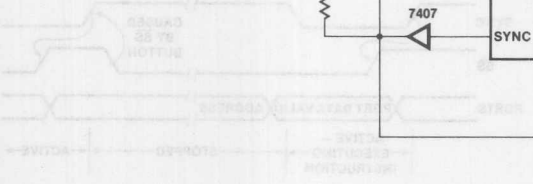


Figure 34. Single Step External Circuitry

LOC	001	002	003	004	005	006	007	008	009	010	011	012	013	014	015	016	017	018	019	020	021	022	023	024	025	026	027	028	029	030	031	032	033	034	035	036	037	038	039	040	041	042	043	044	045	046	047	048	049	050	051	052	053	054	055	056	057	058	059	060	061	062	063	064	065	066	067	068	069	070	071	072	073	074	075	076	077	078	079	080	081	082	083	084	085	086	087	088	089	090	091	092	093	094	095	096	097	098	099	100	101	102	103	104	105	106	107	108	109	110	111	112	113	114	115	116	117	118	119	120	121	122	123	124	125	126	127	128	129	130	131	132	133	134	135	136	137	138	139	140	141	142	143	144	145	146	147	148	149	150	151	152	153	154	155	156	157	158	159	160	161	162	163	164	165	166	167	168	169	170	171	172	173	174	175	176	177	178	179	180	181	182	183	184	185	186	187	188	189	190	191	192	193	194	195	196	197	198	199	200	201	202	203	204	205	206	207	208	209	210	211	212	213	214	215	216	217	218	219	220	221	222	223	224	225	226	227	228	229	230	231	232	233	234	235	236	237	238	239	240	241	242	243	244	245	246	247	248	249	250	251	252	253	254	255	256	257	258	259	260	261	262	263	264	265	266	267	268	269	270	271	272	273	274	275	276	277	278	279	280	281	282	283	284	285	286	287	288	289	290	291	292	293	294	295	296	297	298	299	300	301	302	303	304	305	306	307	308	309	310	311	312	313	314	315	316	317	318	319	320	321	322	323	324	325	326	327	328	329	330	331	332	333	334	335	336	337	338	339	340	341	342	343	344	345	346	347	348	349	350	351	352	353	354	355	356	357	358	359	360	361	362	363	364	365	366	367	368	369	370	371	372	373	374	375	376	377	378	379	380	381	382	383	384	385	386	387	388	389	390	391	392	393	394	395	396	397	398	399	400	401	402	403	404	405	406	407	408	409	410	411	412	413	414	415	416	417	418	419	420	421	422	423	424	425	426	427	428	429	430	431	432	433	434	435	436	437	438	439	440	441	442	443	444	445	446	447	448	449	450	451	452	453	454	455	456	457	458	459	460	461	462	463	464	465	466	467	468	469	470	471	472	473	474	475	476	477	478	479	480	481	482	483	484	485	486	487	488	489	490	491	492	493	494	495	496	497	498	499	500	501	502	503	504	505	506	507	508	509	510	511	512	513	514	515	516	517	518	519	520	521	522	523	524	525	526	527	528	529	530	531	532	533	534	535	536	537	538	539	540	541	542	543	544	545	546	547	548	549	550	551	552	553	554	555	556	557	558	559	560	561	562	563	564	565	566	567	568	569	570	571	572	573	574	575	576	577	578	579	580	581	582	583	584	585	586	587	588	589	590	591	592	593	594	595	596	597	598	599	600	601	602	603	604	605	606	607	608	609	610	611	612	613	614	615	616	617	618	619	620	621	622	623	624	625	626	627	628	629	630	631	632	633	634	635	636	637	638	639	640	641	642	643	644	645	646	647	648	649	650	651	652	653	654	655	656	657	658	659	660	661	662	663	664	665	666	667	668	669	670	671	672	673	674	675	676	677	678	679	680	681	682	683	684	685	686	687	688	689	690	691	692	693	694	695	696	697	698	699	700	701	702	703	704	705	706	707	708	709	710	711	712	713	714	715	716	717	718	719	720	721	722	723	724	725	726	727	728	729	730	731	732	733	734	735	736	737	738	739	740	741	742	743	744	745	746	747	748	749	750	751	752	753	754	755	756	757	758	759	760	761	762	763	764	765	766	767	768	769	770	771	772	773	774	775	776	777	778	779	780	781	782	783	784	785	786	787	788	789	790	791	792	793	794	795	796	797	798	799	800	801	802	803	804	805	806	807	808	809	810	811	812	813	814	815	816	817	818	819	820	821	822	823	824	825	826	827	828	829	830	831	832	833	834	835	836	837	838	839	840	841	842	843	844	845	846	847	848	849	850	851	852	853	854	855	856	857	858	859	860	861	862	863	864	865	866	867	868	869	870	871	872	873	874	875	876	877	878	879	880	881	882	883	884	885	886	887	888	889	890	891	892	893	894	895	896	897	898	899	900	901	902	903	904	905	906	907	908	909	910	911	912	913	914	915	916	917	918	919	920	921	922	923	924	925	926	927	928	929	930	931	932	933	934	935	936	937	938	939	940	941	942	943	944	945	946	947	948	949	950	951	952	953	954	955	956	957	958	959	960	961	962	963	964	965	966	967	968	969	970	971	972	973	974	975	976	977	978	979	980	981	982	983	984	985	986	987	988	989	990	991	992	993	994	995	996	997	998	999	1000	1001	1002	1003	1004	1005	1006	1007	1008	1009	1010	1011	1012	1013	1014	1015	1016	1017	1018	1019	1020	1021	1022	1023	1024	1025	1026	1027	1028	1029	1030	1031	1032	1033	1034	1035	1036	1037	1038	1039	1040	1041	1042	1043	1044	1045	1046	1047	1048	1049	1050	1051	1052	1053	1054	1055	1056	1057	1058	1059	1060	1061	1062	1063	1064	1065	1066	1067	1068	1069	1070	1071	1072	1073	1074	1075	1076	1077	1078	1079	1080	1081	1082	1083	1084	1085	1086	1087	1088	1089	1090	1091	1092	1093	1094	1095	1096	1097	1098	1099	1100	1101	1102	1103	1104	1105	1106	1107	1108	1109	1110	1111	1112	1113	1114	1115	1116	1117	1118	1119	1120	1121	1122	1123	1124	1125	1126	1127	1128	1129	1130	1131	1132	1133	1134	1135	1136	1137	1138	1139	1140	1141	1142	1143	1144	1145	1146	1147	1148	1149	1150	1151	1152	1153	1154	1155	1156	1157	1158	1159	1160	1161	1162	1163	1164	1165	1166	1167	1168	1169	1170	1171	1172	1173	1174	1175	1176	1177	1178	1179	1180	1181	1182	1183	1184	1185	1186	1187	1188	1189	1190	1191	1192	1193	1194	1195	1196	1197	1198	1199	1200	1201	1202	1203	1204	1205	1206	1207	1208	1209	1210	1211	1212	1213	1214	1215	1216	1217	1218	1219	1220	1221	1222	1223	1224	1225	1226	1227	1228	1229	1230	1231	1232	1233	1234	1235	1236	1237	1238	1239	1240	1241	1242	1243	1244	1245	1246	1247	1248	1249	1250	1251	1252	1253	1254	1255	1256	1257	1258	1259	1260	1261	1262	1263	1264	1265	1266	1267	1268	1269	1270	1271	1272	1273	1274	1275	1276	1277	1278	1279	1280	1281	1282	1283	1284	1285	1286	1287	1288	1289	1290	1291	1292	1293	1294	1295	1296	1297	1298	1299	1300	1301	1302	1303	1304	1305	1306	1307	1308	1309	1310	1311	1312	1313	1314	1315	1316	1317	1318	1319	1320	1321	1322	1323	1324	1325	1326	1327	1328	1329	1330	1331	1332	1333	1334	1335	1336	1337	1338	1339	1340	1341	1342	1343	1344	1345	1346	1347	1348	1349	1350	1351	1352	1353	1354	1355	1356	1357	1358	1359	1360	1361	1362	1363	1364	1365	1366	1367	1368	1369	1370	1371	1372	1373	1374	1375	1376	1377	1378	1379	1380	1381	1382	1383	1384	1385	1386	1387	1388	1389	1390	1391	1392	1393	1394	1395	1396	1397	1398	1399	1400	1401	1402	1403	1404	1405	1406	1407	1408	1409	1410	1411	1412	1413	1414	1415	1416	1417	1418	1419	1420	1421	1422	1423	1424	1425	1426	1427	1428	1429	1430	1431	1432	1433	1434	1435	1436	1437	1438	1439	1440	1441	1442	1443	1444	1445	1446	1447	1448	1449	1450	1451	1452	1453	1454	1455	1456	1457	1458	1459	1460	1461	1462	1463	1464	1465	1466	1467	1468	1469	1470	1471
-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------

APPENDIX A1

IS15-II MCS-48/UIP-41 MACRO ASSEMBLER, V2.0

LOC	OBJ	SEQ	SOURCE STATEMENT						
1			*****						
2			* UIP-41 8-DIGIT LED DISPLAY CONTROLLER *						
3			*****						
4									
5									
6									
7			THIS PROGRAM USES THE UIP-41 AS A LED DISPLAY CONTROLLER						
8			WHICH SCANS AND REFRESHES EIGHT SEVEN-SEGMENT LED DISPLAYS.						
9			THE CHARACTERS ARE DEFINED BY INPUT FROM A MASTER CPU IN THE						
10			FORM OF ONE EIGHT BIT WORD PER DIGIT-CHARACTER SELECTION.						
11									
12									
13									
14			*****						
15									
16			REGISTER DEFINITIONS:						
17			<table border="0"> <tr> <th>REGISTER</th> <th>RB1</th> <th>RB0</th> </tr> <tr> <td>---</td> <td>---</td> <td>---</td> </tr> </table>	REGISTER	RB1	RB0	---	---	---
REGISTER	RB1	RB0							
---	---	---							
18									
19			R0 DISPLAY MAP POINTER NOT USED						
20			R1 NOT USED NOT USED						
21			R2 DATA WORD AND CHARACTER STORAGE NOT USED						
22			R3 DIGIT COUNTER NOT USED						
23			R4 NOT USED NOT USED						
24			R5 NOT USED NOT USED						
25			R6 NOT USED NOT USED						
26			R7 ACCUMULATOR STORAGE NOT USED						
27			*****						
28									
29			PORT PIN DEFINITIONS:						
30			<table border="0"> <tr> <th>PIN</th> <th>PORT 1 FUNCTION</th> <th>PORT 2 FUNCTION</th> </tr> <tr> <td>---</td> <td>---</td> <td>---</td> </tr> </table>	PIN	PORT 1 FUNCTION	PORT 2 FUNCTION	---	---	---
PIN	PORT 1 FUNCTION	PORT 2 FUNCTION							
---	---	---							
31									
32			P0-7 SEGMENT DRIVER CONTROL DIGIT DRIVER CONTROL						
33									
34			\$EJECT						

APPENDIX A1 (Continued)

LOC	OBJ	SEQ	SOURCE STATEMENT
35			*****
36			DISPLAY DATA WORD BIT DEFINITION:
37			BIT FUNCTION
38			-----
39			0-4 CHARACTER SELECT
40			5-7 DIGIT SELECT
41			
42			CHARACTER SELECT:
43			D4 D3 D2 D1 D0 CHARACTER
44			0 0 0 0 0 0
45			0 0 0 0 0 1
46			0 0 0 0 1 0
47			0 0 0 0 1 1
48			0 0 1 0 0 0
49			0 0 1 0 0 1
50			0 0 1 1 0 0
51			0 0 1 1 1 0
52			0 1 0 0 0 0
53			0 1 0 0 0 1
54			0 1 0 1 0 0
55			0 1 0 1 0 1
56			0 1 1 0 0 0
57			0 1 1 0 0 1
58			0 1 1 1 0 0
59			0 1 1 1 1 0
60			1 0 0 0 0 0
61			1 0 0 0 0 1
62			1 0 0 1 0 0
63			1 0 0 1 0 1
64			1 0 1 0 0 0
65			1 0 1 0 0 1
66			1 0 1 1 0 0
67			1 0 1 1 0 1
68			1 1 0 0 0 0
69			1 1 0 0 0 1
70			1 1 0 1 0 0
71			1 1 0 1 0 1
72			1 1 1 0 0 0
73			1 1 1 0 0 1
74			1 1 1 1 0 0
75			1 1 1 1 0 1 "BLANK"
76			
77			
78			DIGIT SELECT:
79			D7 D6 D5 DIGIT NUMBER
80			0 0 0 1
81			0 0 1 2
82			0 1 0 3
83			0 1 1 4
84			1 0 0 5
85			1 0 1 6
86			1 1 0 7
87			1 1 1 8
88			*****
89			\$EJECT

APPENDIX A1 (Continued)

LOC	OBJ	SEQ	SOURCE STATEMENT
		90	*****
		91	EQUATES
		92	THE FOLLOWING CODE DESIGNATES "TIME" AS A VARIABLE. THIS
		93	ADJUSTS THE AMOUNT OF CYCLES THE TIMER COUNTS BEFORE
		94	A TIMER INTERRUPT OCCURS AND REFRESHES THE DISPLAY. APPROXIMATELY
		95	50 TIMES PER SECOND.
FFF1		96	TIME EQU -0FH ;TIMER VALUE 2.5MSEC
		97	*****
		98	INTERRUPT BRANCHING
		99	THIS PORTION OF MEMORY IS DEDICATED FOR USE OF RESET AND
		100	INTERRUPT BRANCHING WHEN THE INTERRUPTS ARE ENABLED THE
		101	CODE AT THE FOLLOWING DESIGNATED SPOTS ARE EXECUTED WHEN A
		102	RESET OR A INTERRUPT OCCURS.
0000		103	ORG 0 ;
0000 0409		104	JMP START ;RESET
0002 00		105	NOP ;
0003 0438		106	JMP INPUT ;IBF INTERRUPT
0005 00		107	NOP ;
0006 00		108	NOP ;
0007 041F		109	JMP DISPLA ;TIMER INTERRUPT
		110	*****
		111	INITIALIZATION
		112	THE FOLLOWING CODE SETS UP THE UPI-41 AND DISPLAY HARDWARE
		113	INTO OPERATIONAL FORMAT. THE DISPLAY IS TURNED OFF, THE DISPLAY
		114	MAP IS FILLED WITH "BLANK" CHARACTERS, THE TIMER SET AND THE
		115	INTERRUPTS ARE ENABLED.
		116	;
0009 05		117	START: SEL R01 ;
000A 0A08		118	ORL P2, #00H ;TURN DIGIT DRIVERS OFF
000C B838		119	MOV R0, #38H ;DISPLAY MAP POINTER, BOTTOM OF DISPLAY MAP
000E 23FF		120	BLKMAP: MOV A, #0FFH ;FF="BLANK"
0010 A0		121	MOV @R0, A ;BLANK TO DISPLAY MAP
0011 18		122	INC R0 ;INCREMENT DISPLAY MAP POINTER
0012 F8		123	MOV A, R0 ;DISPLAY MAP POINTER TO ACCUMULATOR
0013 B20E		124	JB5 BLKMAP ;BLANK DISPLAY MAP TILL FILLED
0015 BB00		125	MOV R3, #00H ;SET DIGIT COUNTER TO 0
0017 23F1		126	MOV A, #TIME ;TIMER VALUE
0019 62		127	MOV T, A ;LOAD TIMER
001A 55		128	STRT T ;START TIMER
001B 25		129	EN TCNTI ;ENABLE TIMER INTERRUPT
001C 05		130	EN I ;ENABLE IBF INTERRUPT
		131	*****
		132	USER PROGRAM
		133	A USERS PROGRAM WOULD INITIALIZE AT THIS POINT. THE FOLLOWING
		134	CODE IS USED TO TAKE THE PLACE OF A POSSIBLE USER PROGRAM.
		135	;
		136	;
001D 041D		137	LOOP: JMP LOOP ;WAIT FOR INTERRUPT
		138	*****
		139	\$EJECT

APPENDIX A1 (Continued)

LOC	OBJ	SEQ	SOURCE STATEMENT
		140	;*****
		141	; DISPLAY ROUTINE
		142	; THIS PORTION OF THIS PROGRAM IS AN INTERRUPT ROUTINE WHICH IS
		143	;ACTED UPON WHEN THE TIMER COUNT IS COMPLETED. THE ROUTINE UPDATES
		144	;ONE DISPLAY DIGIT FROM THE DISPLAY MAP PER INTERRUPT SEQUENTIALLY,
		145	;THUS EIGHT TIMER INTERRUPTS WILL HAVE REFRESHED THE ENTIRE DISPLAY.
		146	;REGISTER BANK 1 IS SELECTED AND THE ACCUMULATOR IS SAVED UPON
		147	;ENTERING THE ROUTINE. ONCE THE DISPLAY HAS BEEN REFRESHED THE TIMER
		148	;IS RESET AND THE ACCUMULATOR AND PRE-INTERRUPT REGISTER BANK IS RESTORED.
		149	;*****
001F	D5	150	DISPLA: SEL R01 ;REGISTER BANK 1
0020	AF	151	MOV R7,A ;SAVE ACCUMULATOR
0021	8A08	152	ORL P2,#08H ;TURN DIGIT DRIVERS OFF
0023	FB	153	MOV A,R3 ;DIGIT COUNTER TO ACCUMULATOR
0024	4338	154	ORL A,#38H ;"OR" TO GET DISPLAY MAP ADDRESS
0026	A8	155	MOV R0,A ;DISPLAY MAP POINTER
0027	F0	156	MOV A,@R0 ;GET CHARACTER FROM DISPLAY MAP
0028	39	157	OUTL P1,A ;OUTPUT CHARACTER TO SEGMENT DRIVERS
0029	FB	158	MOV A,R3 ;DIGIT COUNTER VALUE TO ACCUMULATOR
002A	3A	159	OUTL P2,A ;OUTPUT TO DIGIT DRIVERS
002B	1B	160	INC R3 ;INCREMENT DIGIT COUNTER
002C	D307	161	XRL A,#07H ;CHECK IF AT LAST DIGIT
002E	9632	162	JNZ SETIME ;RESET TIMER IN NOT LAST DIGIT
0030	B000	163	MOV R3,#00H ;RESET DIGIT COUNTER
0032	23F1	164	SETIME: MOV A,#TIME ;TIMER VALUE
0034	62	165	MOV T,A ;LOAD TIMER
0035	55	166	STRT T ;START TIMER
0036	FF	167	MOV A,R7 ;RESTORE ACCUMULATOR
0037	93	168	RETR ;RETURN
		169	;*****
		170	\$EJECT

APPENDIX A1 (Continued)

LOC	OBJ	SEQ	SOURCE STATEMENT	THIRTYTWO 30302	302	003
		171	;			
		172	*****			
		173	INPUT CHARACTER AND DIGIT ROUTINE			
		174	THIS PORTION OF THE PROGRAM IS AN INTERRUPT ROUTINE WHICH			
		175	IS ACTED UPON WHEN THE IBF BIT IS SET. THE ROUTINE GETS THE			
		176	DISPLAY DATA WORD FROM THE DBB AND DEFINES BOTH THE DIGIT AND			
		177	THE CHARACTER TO BE DISPLAYED. THIS IS DONE BY MEANS OF A			
		178	CHARACTER LOOP-UP TABLE AND A DISPLAY MAP FOR DIGIT AND CHARACTER			
		179	LOCATION. SPECIAL CONSIDERATION IS TAKEN FOR A DECIMAL POINT WHICH IS			
		180	SIMPLY ADDED TO THE EXISTING CHARACTER IN THE DISPLAY MAP. REGISTER			
		181	BANK 1 IS SELECTED AND THE ACCUMULATOR IS SAVED UPON ENTERING			
		182	THE ROUTINE. ONCE THE DATA WORD HAS BEEN FULLY DEFINED THE ACCUMULATOR			
		183	AND THE PRE-INTERRUPT REGISTER BANK IS RESTORED.			
		184	;			
0038	D5	185	INPUT: SEL R01 ; REGISTER BANK 1			
0039	AF	186	MOV R7, A ; SAVE ACCUMULATOR			
003A	22	187	IN A, DBB ; GET DATA			
003B	AA	188	MOV R2, A ; SAVE DATA WORD			
003C	47	189	SWAP A ; DEFINE DIGIT LOCATION			
003D	77	190	RR A ;			
003E	5307	191	ANL A, #07H ;			
0040	4338	192	ORL A, #38H ;			
0042	A8	193	MOV R0, A ; DIGIT LOCATION IN DIGIT POINTER			
0043	FA	194	MOV A, R2 ; SAVED DATA WORD TO ACCUMULATOR			
0044	531F	195	ANL A, #1FH ; DEFINE CHARACTER LOOK-UP-TABLE LOC.			
0046	E3	196	MOVP3 A, 0A ; GET CHARACTER			
0047	AA	197	MOV R2, A ; SAVE CHARACTER			
0048	D37F	198	XRL A, #7FH ; IS CHARACTER DECIMAL POINT			
004A	C650	199	JZ DPOINT ;			
004C	FA	200	MOV A, R2 ; SAVED CHARACTER TO ACCUMULATOR			
004D	A0	201	MOV @R0, A ; CHARACTER TO DISPLAY MAP			
004E	0453	202	JMP RETURN ;			
0050	FA	203	DPOINT: MOV A, R2 ; SAVED CHARACTER TO ACCUMULATOR			
0051	50	204	ANL A, @R0 ; "AND" WITH OLD CHARACTER			
0052	A0	205	MOV @R0, A ; BACK TO DISPLAY MAP			
0053	FF	206	RETURN: MOV A, R7 ; RESTORE ACCUMULATOR			
0054	93	207	RETR			
		208	*****			
		209	\$EJECT			

APPENDIX A1 (Continued)

LOC	OBJ	SEQ	SOURCE STATEMENT
210			*****
211			LOOK-UP TABLE
212			THIS LOOK-UP TABLE ORIGINATES IN PAGE 3 OF THE UPI-41 PROGRAM
213			MEMORY. IT IS USED TO DEFINE THE CORRECT LEVEL OF EACH SEGMENT
214			AND DECIMAL POINT FOR A SELECTED CHARACTER FROM THE INPUT ROUTINE.
215			INVERSE LOGIC IS USED BECAUSE OF THE SPECIFIC DRIVER CIRCUITRY, THUS
216			A 1 ON A GIVEN SEGMENT MEANS IT IS OFF AND A 0 MEANS IT IS ON.
217			
218			*****SEGMENTS*****
0300		219	ORG 300H ;DP G F E D C B A
0300 C0		220 CH0: DB 0C0H ;1 1 0 0 0 0 0 0	
0301 F9		221 CH1: DB 0F9H ;1 1 1 1 1 0 0 1	
0302 A4		222 CH2: DB 0A4H ;1 0 1 0 0 1 0 0	
0303 B0		223 CH3: DB 0B0H ;1 0 1 1 0 0 0 0	
0304 99		224 CH4: DB 99H ;1 0 0 1 1 0 0 1	
0305 92		225 CH5: DB 92H ;1 0 0 1 0 0 1 0	
0306 82		226 CH6: DB 82H ;1 0 0 0 0 0 1 0	
0307 F8		227 CH7: DB 0F8H ;1 1 1 1 1 0 0 0	
0308 80		228 CH8: DB 80H ;1 0 0 0 0 0 0 0	
0309 98		229 CH9: DB 98H ;1 0 0 1 1 0 0 0	
030A 88		230 CHA: DB 88H ;1 0 0 0 1 0 0 0	
030B 83		231 CHB: DB 83H ;1 0 0 0 0 0 1 1	
030C C6		232 CHC: DB 0C6H ;1 1 0 0 0 1 1 0	
030D A1		233 CHD: DB 0A1H ;1 0 1 0 0 0 0 1	
030E 86		234 CHE: DB 86H ;1 0 0 0 0 1 1 0	
030F 8E		235 CHF: DB 8EH ;1 0 0 0 1 1 1 0	
0310 7F		236 CHDP: DB 7FH ;0 1 1 1 1 1 1 1	
0311 C2		237 CHG: DB 0C2H ;1 1 0 0 0 0 1 0	
0312 89		238 CHH: DB 89H ;1 0 0 0 1 0 0 1	
0313 FB		239 CHI: DB 0FBH ;1 1 1 1 1 0 1 1	
0314 E1		240 CHJ: DB 0E1H ;1 1 1 0 0 0 0 1	
0315 C7		241 CHL: DB 0C7H ;1 1 0 0 0 1 1 1	
0316 AB		242 CHN: DB 0ABH ;1 0 1 0 1 0 1 1	
0317 A3		243 CHO: DB 0A3H ;1 0 1 0 0 0 1 1	
0318 8C		244 CHP: DB 8CH ;1 0 0 0 1 1 0 0	
0319 AF		245 CHR: DB 0AFH ;1 0 1 0 1 1 1 1	
031A 87		246 CHT: DB 87H ;1 0 0 0 0 1 1 1	
031B C1		247 CHU: DB 0C1H ;1 1 0 0 0 0 0 1	
031C 91		248 CHV: DB 91H ;1 0 0 1 0 0 0 1	
031D BF		249 CHDASH: DB 0BFH ;1 0 1 1 1 1 1 1	
031E FD		250 CHAPOS: DB 0FDH ;1 1 1 1 1 1 0 1	
031F FF		251 BLANK: DB 0FFH ;1 1 1 1 1 1 1 1	
252			*****
253			END
USER SYMBOLS			
BLANK	031F	BLKMAP	000E CH0 0300 CH1 0301 CH2 0302 CH3 0303 CH4 0304 CH5 0305
CH6	0306	CH7	0307 CH8 0308 CH9 0309 CHA 030A CHAPOS 031E CHB 030B CHC 030C
CHD	030D	CHDASH	031D CHDP 0310 CHE 030E CHF 030F CHG 0311 CHH 0312 CHI 0313
CHJ	0314	CHL	0315 CHN 0316 CHO 0317 CHP 0318 CHR 0319 CHT 031A CHU 031B
CHV	031C	DISPLA	001F DPOINT 0050 INPUT 0038 LOOP 001D RETURN 0053 SETIME 0032 START 0009
TIME	FFF1		

ASSEMBLY COMPLETE, NO ERRORS

APPENDIX A2

LOC	OBJ	SEQ	SOURCE STATEMENT
		1 ;	
		2 ;	8085A SUBROUTINE TO DISPLAY THE 8-DIGIT BUFFER STARTING
		3 ;	AT THE LOCATION POINTED AT BY MSGSRT ON THE UPI-CONTROLLED
		4 ;	LED DISPLAY.
		5 ;	
		6 ;	INPUTS: MSGSRT - MESSAGE START LOCATION POINTER
		7 ;	DESTROYS: A, F/F'S
		8 ;	CALLS: OUTCHR
		9 ;	
4000		10	ORG 4000H
00E5		11	STATUS EQU 0E5H ;UPI STATUS PORT
0002		12	IBF EQU 02H ;UPI IBF FLAG MASK
00E4		13	DBBIN EQU 0E4H ;UPI DBBIN PORT
		14 ;	
4000 E5		15	DISPLAY: PUSH H ;SAVE HL
4001 C5		16	PUSH B ;SAVE BC
4002 2A2840		17	LHLD MSGSRT ;LOAD HL WITH MESSAGE START ADR
4005 0600		18	MVI B,00H ;INITIALIZE DIGIT COUNTER
4007 7E		19	S1: MOV A,M ;GET CHR FROM BUFFER
4008 E61F		20	ANI 1FH ;MAKE IT 5 BITS
400A 80		21	ADD B ;ADD IN DIGIT COUNTER
400B 4F		22	MOV C,A ;SAVE TOTAL IN C
400C CD1D40		23	CALL OUTCHR ;OUTPUT CHR PLUS LOCATION TO UPI
400F 78		24	MOV A,B ;GET DIGIT COUNTER
4010 C620		25	ADI 20H ;INC FOR NEXT DIGIT
4012 DA1A40		26	JC EXIT ;DONE IF CARRY SET
4015 47		27	MOV B,A ;RESTORE DIGIT COUNTER
4016 23		28	INX H ;INC MESSAGE POINTER
4017 C30740		29	JMP S1 ;GO GET NEXT CHR
		30 ;	
401A C1		31	EXIT: POP B ;RESTORE BC
401B E1		32	POP H ;RESTORE HL
401C C9		33	RET ;RETURN
		34 ;	
		35 ;	SUBROUTINE TO OUTPUT CHR TO UPI
		36 ;	
401D DBE5		37	OUTCHR: IN STATUS ;READ UPI STATUS
401F E602		38	ANI IBF ;LOOK AT IBF
4021 C21D40		39	JNZ OUTCHR ;WAIT UNTIL IBF=0
4024 79		40	MOV A,C ;GET CHR
4025 D3E4		41	OUT DBBIN ;OUTPUT CHR TO UPI DBBIN
4027 C9		42	RET ;RETURN
		43 ;	
0002		44	MSGSRT: DS 02H ;LOCATION OF MESSAGE START POINTER
		45 ;	
		46	END


```

LOC  OBJ      SEQ      SOURCE STATEMENT
1 ;          *****
2 ;          *      UIP-41A SENSOR MATRIX CONTROLLER      *
3 ;          *****
4 ;
5 ;          THIS PROGRAM USES THE UIP-41A AS A SENSOR MATRIX CONTROLLER.
6 ; IT HAS MONITORING CAPABILITIES OF UP TO 128 SENSORS.  THE COORDINATE
7 ; AND SENSOR STATUS OF EACH DETECTED CHANGE IS AVAILABLE TO THE MASTER
8 ; MICROPROCESSOR IN A SINGLE BYTE.  A 40X8 FIFO QUEUE IS PROVIDED FOR
9 ; DATA BUFFERING.  BOTH HARDWARE OR POLLED INTERRUPT METHODS CAN BE USED
10 ; TO NOTIFY THE MASTER OF A DETECTED SENSOR CHANGE.
11 ;
12 ; *****
13 ;
14 ; REGISTER DEFINITIONS:
15 ;      REGISTER      RBQ      RBI
16 ;      -----
17 ;      R0      MATRIX MAP POINTER      NOT USED
18 ;      R1      FIFO POINTER      NOT USED
19 ;      R2      SCAN ROW SELECT      NOT USED
20 ;      R3      COLUMN COUNTER      NOT USED
21 ;      R4      FIFO-IN      NOT USED
22 ;      R5      FIFO-OUT      NOT USED
23 ;      R6      CHANGE WORD      NOT USED
24 ;      R7      COMPARE      NOT USED
25 ;
26 ; *****
27 ;
28 ; PORT PIN DEFINITIONS:
29 ;
30 ; PIN      PORT 1 FUNCTION      PIN      PORT 2 FUNCTION
31 ;      -----
32 ; P0-7      COLUMN LINE INPUTS      P0-3      ROW SELECT OUTPUTS
33 ;
34 ;
35 ;
36 ;
37 ; *****
38 ;
39 $EJECT

```

APPENDIX B1 (Continued)

```

43 ;
44 ;          BIT          FUNCTION
45 ;          ---          -
46 ;          D0-6          SENSOR COORDINATE
47 ;          D7            SENSOR STATUS
48 ;
49 ;*****
50 ;
51 ;STATUS REGISTER BIT DEFINITION:
52 ;
53 ;          BIT          FUNCTION
54 ;          ---          -
55 ;          D0            OBF
56 ;          D1-3          IBF, F0, F1 (NOT USED)
57 ;          D4            FIFO NOT EMPTY
58 ;          D5-7          USED DEFINED (NOT USED)
59 ;
60 ;*****
61 ;
62 ;          EQUATES
63 ;
64 ;THE FOLLOWING CODE DESIGNATES THREE VARIABLES; SCANTM, FIF0BA
65 ;AND FIF0TA. SCANTM ADJUSTS THE LENGTH OF A DELAY BETWEEN
66 ;SCANNING SWITCH. THIS SIMULATES DEBOUNCE FUNCTIONS. FIF0BA
67 ;IS THE BOTTOM ADDRESS OF THE FIFO. FIF0TA IS THE TOP ADDRESS
68 ;OF THE FIFO. THIS MAKES IT POSSIBLE TO HAVE A FIFO 3 TO 40
69 ;BYTES IN LENGTH.
70 ;
71 ;*****
72 ;
000F 73 SCANTM EQU 0FH          ;SCAN TIME ADJUST
0008 74 FIF0BA EQU 08H          ;FIFO BOTTOM ADDRESS
002F 75 FIF0TA EQU 2FH          ;FIFO TOP ADDRESS
76 ;
77 $EJECT

```


APPENDIX B1 (Continued)

LOC	OBJ	SEQ	SOURCE STATEMENT
		78 ;	*****
		79 ;	
		80 ;	INITIALIZATION
		81 ;	
		82 ;	THE PROGRAM STARTS AT THE FOLLOWING CODE UPON RESET. WITHIN
		83 ;	THIS INITIALIZATION SECTION THE REGISTERS THAT MAINTAIN THE MATRIX
		84 ;	MAP, FIFO AND ROW SCANNING ARE SET UP. PORT 1 IS SET HIGH FOR USE
		85 ;	AS AN INPUT PORT FOR THE COLUMN STATUS. BIT 4 OF STATUS REGISTER IS
		86 ;	WRITTEN TO CONVEY A FIFO EMPTY CONDITION. THE INITIAL COLUMN STATUS
		87 ;	OF ALL THE ROWS IN THE SENSOR MATRIX IS THEN READ INTO THE MATRIX
		88 ;	MAP. ONCE THE MATRIX MAP IS FILLED THE OBF INTERRUPT (PORT 2-4) IS
		89 ;	ENABLED.
		90 ;	
		91 ;	*****
		92 ;	
0000		93	ORG 0
0000 B83F		94 INITMX:	MOV R0, #3FH ; MATRIX MAP POINTER REGISTER, TOP ADDRESS
0002 BA0F		95	MOV R2, #0FH ; SCAN ROW SELECT REGISTER, TOP ROW
0004 BC08		96	MOV R4, #FIFOBA ; FIFO INPUT ADDRESS REGISTER, BOTTOM OF FIFO
0006 BD2F		97	MOV R5, #FIFOTA ; FIFO OUTPUT ADDRESS REGISTER, TOP OF FIFO
0008 89FF		98	ORL P1, #0FFH ; INITIALIZE PORT 1 HIGH FOR INPUTS
000A 2300		99	MOV A, #00H ; INITIALIZE STATUS REGISTER, FIFO EMPTY
000C 90	100	MOV	ST5, A ; WRITE TO STATUS REGISTER, BITS 4-7
000D FA	101 FILLMX:	MOV	A, R2 ; SCAN ROW SELECT TO ACCUMULATOR
000E 3A	102	OUTL	P2, A ; OUTPUT SCAN ROW SELECT TO PORT 2
000F 09	103	IN	A, P1 ; INPUT COLUMN STATUS PORT 1
0010 A0	104	MOV	@R0, A ; LOAD MATRIX MAP WITH COLUMN STATUS
0011 FA	105	MOV	A, R2 ; CHECK SCAN ROW SELECT REGISTER VALUE FOR 0
0012 C618	106	JZ	OBFINT ; IF 0 ENABLE OBF INTERRUPT
0014 C8	107	DEC	R0 ; DECREMENT TO NEXT MATRIX MAP ADDRESS
0015 CA	108	DEC	R2 ; DECREMENT TO SCAN NEXT ROW
0016 040D	109	JMP	FILLMX ; FILL NEXT MATRIX MAP ADDRESS
0018 BA10	110 OBFINT:	MOV	R2, #10H ; BIT 4 HIGH IN ROW SCAN SELECT REGISTER
001A FA	111	MOV	A, R2 ; ROW SCAN SELECT VALUE TO ACCUMULATOR
001B 3A	112	OUTL	P2, A ; INITIALIZE PORT 2, BIT 4 FOR "EN FLAGS"
001C F5	113	EN	FLAGS ; ENABLE OBF INTERRUPT PORT 2, BIT 4
		114 ;	
		115	\$EJECT

APPENDIX B1 (Continued)

LOC	OBJ	SEQ	SOURCE STATEMENT
		116	*****
		117	;
		118	SCAN AND COMPARE
		119	;
		120	THE FOLLOWING CODE IS THE SCAN AND COMPARE SECTION OF THE PROGRAM.
		121	UPON ENTERING THIS SECTION A CHECK IS MADE TO SEE IF THE ENTIRE MATRIX
		122	HAS BEEN SCANNED. IF SO THE REGISTERS THAT MAINTAIN THE MATRIX MAP AND ROW
		123	SCANNING ARE RESET TO THE BEGINNING OF THE SENSOR MATRIX. IF THE ENTIRE
		124	MATRIX HASNT BEEN SCANNED THE REGISTERS INCREMENT TO SCAN THE NEXT ROW.
		125	FROM THIS POINT ON THE ROW SCAN SELECT REGISTER IS USED FOR TWO FUNCTIONS.
		126	BITS 0-3 FOR SCANNING AND BITS 4 AND 5 FOR THE EXTERNAL INTERRUPTS. THUSLY
		127	ALL USAGE OF THE REGISTERS IS DONE BY LOGICALLY MASKING IT SO AS TO ONLY
		128	AFFECT THE FUNCTION DESIRED. ONCE THE REGISTERS ARE RESET, ONE ROW OF THE
		129	SENSOR MATRIX IS SCANNED. A DELAY IS EXECUTED TO ADJUST FOR SCAN TIME
		130	(DEBOUNCE). A BYTE OF COLUMN STATUS IS THEN READ INTO THE MATRIX MAP.
		131	AT THE TIME THE NEW COLUMN STATUS IS COMPARED TO THE OLD. THE RESULT IS
		132	STORED IN THE COMPARE REGISTER. THE PROGRAM IS THEN ROUTED ACCORDING TO
		133	WHETHER OR NOT A CHANGE WAS DETECTED.
		134	;
		135	*****
		136	;
001D	FA	137	ADJREG: MOV A, R2 ; SCAN ROW SELECT TO ACCUMULATOR
001E	530F	138	ANL A, #0FH ; CHECK FOR 0 SCAN VALUE ONLY, NOT INTERRUPT
0020	C626	139	JZ RSETRG ; IF 0 RESET REGISTERS
0022	C8	140	DEC R0 ; DECREMENT MATRIX MAP POINTER
0023	CA	141	DEC R2 ; DECREMENT SCAN ROW SELECT
0024	042C	142	JMP SCANMX ; SCAN MATRIX
0026	B83F	143	RSETRG: MOV R0, #3FH ; RESET MATRIX MAP POINTER REGISTER, TOP ADDRESS
0028	FA	144	MOV A, R2 ; SCAN ROW SELECT TO ACCUMULATOR
0029	430F	145	ORL A, #0FH ; RESET SCAN ROW SELECT, NO INTERRUPT CHANGE
002B	AA	146	MOV R2, A ; SCAN ROW SELECT REGISTER
002C	FA	147	SCANMX: MOV A, R2 ; SCAN ROW SELECT TO ACCUMULATOR
002D	3A	148	OUTL P2, A ; OUTPUT SCAN ROW SELECT TO PORT 2
002E	B80F	149	MOV R3, #SCANTM ; SET DELAY FOR OUTPUT SCAN TIME
0030	EB30	150	DELAY2: DJNZ R3, DELAY2 ; DELAY
0032	09	151	IN A, P1 ; INPUT COLUMN STATUS FROM PORT 1 TO ACCUMULATOR
0033	20	152	XCH A, @R0 ; STORE NEW COLUMN STATUS SAVE OLD IN ACCUMULATOR
0034	D0	153	XRL A, @R0 ; COMPARE OLD WITH NEW COLUMN STATUS
0035	AF	154	MOV R7, A ; SAVE COMPARE RESULT IN COMPARE REGISTER
0036	C669	155	JZ CHFFUL ; IF THE SAME, CHECK IF FIFO IS FULL
		156	;
		157	\$EJECT

APPENDIX B1 (Continued)

LOC	OBJ	SEQ	SOURCE STATEMENT	DISPATCH	STATUS	LOC	OBJ
		158	*****				
		159	;				
		160	CHANGE WORD ENCODING				
		161	;				
		162	THE FOLLOWING CODE IS THE CHANGE WORD ENCODING SECTION. THIS				
		163	SECTION IS ONLY EXECUTED IF A CHANGE WAS DETECTED. THE COLUMN COUNTER				
		164	IS SET AND DECREMENTED TO DESIGNATE EACH OF THE 8 COLUMNS. THE COMPARE				
		165	REGISTER IS LOOKED AT ONE BIT AT A TIME TO FIND THE EXACT LOCATION OF				
		166	THE CHANGE(S). WHEN A CHANGE IS FOUND IT IS ENCODED BY GIVING IT A				
		167	COORDINATE FOR ITS LOCATION. THIS IS DONE BY COMBINING THE PRESENT VALUE				
		168	IN THE ROW SCAN SELECT REGISTER AND THE COLUMN COUNTER. THE ACTUAL STATUS				
		169	OF THAT SENSOR IS ESTABLISHED BY LOOKING AT THE CORRESPONDING BYTE IN				
		170	THE MATRIX MAP. THIS STATUS IS COMBINED WITH THE COORDINATE TO ESTABLISH				
		171	THE CHANGE WORD. THE CHANGE WORD IS THEN STORED IN THE CHANGE WORD REGISTER.				
		172	;				
		173	*****				
		174	;				
0038	BB08	175	MOV R3,#08H ;SET COLUMN COUNTER REGISTER TO 8				
003A	CB	176	RRLOOK: DEC R3 ;DECREMENT COLUMN COUNTER				
003B	F0	177	MOV A,@R0 ;COLUMN STATUS TO ACCUMULATOR				
003C	77	178	RR A ;ROTATE COLUMN STATUS RIGHT				
003D	A0	179	MOV @R0,A ;ROTATED COLUMN STATUS BACK TO MATRIX MAP				
003E	FF	180	MOV A,R7 ;COMPARE REGISTER VALUE TO ACCUMULATOR				
003F	77	181	RR A ;ROTATE COMPARE VALUE RIGHT				
0040	AF	182	MOV R7,A ;ROTATED COMPARE VALUE TO COMPARE REGISTER				
0041	F245	183	JB7 ENCODE ;TEST BIT 7 IF CHANGE DETECTED ENCODE CHANGE WORD				
0043	0469	184	JMP CHFFUL ;IF NO CHANGE IS DETECTED CHECK FOR FIFO FULL				
0045	FA	185	ENCODE: MOV A,R2 ;SCAN ROW SELECT TO ACCUMULATOR 0000XXX				
0046	530F	186	ANL A,#0FH ;ROTATE ONLY SCAN VALUE				
0048	E7	187	RL A ;ROTATE LEFT 00000000				
0049	E7	188	RL A ;ROTATE LEFT 00000000				
004A	E7	189	RL A ;ROTATE LEFT 00000000				
004B	4B	190	ORL A,R3 ;ESTABLISH MATRIX COORDINANT 00000000				
		191	;(OR) COLUMN COUNTER VALUE WITH ACCUMULATOR				
004C	AE	192	MOV R6,A ;SAVE COORDINANT IN CHANGE WORD REGISTER				
004D	F0	193	MOV A,@R0 ;COLUMN STATUS FROM MATRIX MAP TO ACCUMULATOR				
004E	5380	194	ANL A,#80H ;0 ALL BITS BUT BIT 7				
0050	4E	195	ORL A,R6 ;(OR) SENSOR STATUS WITH COORDINATE FOR COMPLETED CHANGE WORD				
0051	AE	196	MOV R6,A ;SAVE CHANGE WORD XXXXXXXX				
		197	;				
		198	\$EJECT				

APPENDIX B1 (Continued)

```

200 ;
201 ;           FIFO-DBBOUT MANAGEMENT
202 ;
203 ;THE FOLLOWING CODE IS THE FIFO-DBBOUT MANAGEMENT SECTION OF THE
204 ;PROGRAM. THIS SECTION TAKES AN ENCODED CHANGE WORD AND LOADS IT INTO
205 ;THE FIFO. THE FIFO NOT EMPTY INTERRUPT IS THEN SET AND THE FIFO-IN
206 ;POINTER GETS UPDATED. A FIFO FULL CONDITION IS THEN CHECKED FOR AND
207 ;ROUTED ACCORDINGLY. IF BOTH THE FIFO AND OBF HAVE CHANGE WORDS THE
208 ;PROGRAM LOCKS UP UNTIL THIS HAS CHANGED. IF THE FIFO ISNT FULL COLUMN
209 ;COUNTER= 0, FIFO EMPTY AND OBF CONDITIONS ARE CHECKED. THE FIFO-OUT
210 ;POINTER IS SET AND DBBOUT IS LOADED IF THE FIFO ISNT EMPTY AND OBF ISNT
211 ;SET. IF THIS ISNT THE SITUATION, PROGRAM FLOW IS ROUTED BACK TO THE
212 ;THE SCAN AND COMPARE SECTION TO SCAN THE NEXT ROW.
213 ;
214 ;*****
215 ;
0052 FC      216 LOADFF: MOV    A,R4      ;FIFO INPUT ADDRESS TO ACCUMULATOR
0053 A9      217      MOV    R1,A      ;FIFO POINTER USED FOR INPUT
0054 FE      218      MOV    A,R6      ;CHANGE WORD TO ACCUMULATOR
0055 A1      219      MOV    @R1,A      ;LOAD FIFO AT FIFO INPUT ADDRESS
0056 2310     220 STATNE: MOV    A,#10H    ;BIT 4 FOR FIFO NOT EMPTY
0058 90      221      MOV    STS,A      ;WRITE TO STATUS REGISTER, FIFO NOT EMPTY
0059 8A20     222 INTRH1: ORL    P2,#20H    ;FIFO NOT EMPTY INTERRUPT PORT 2-5 HIGH
005B FA      223      MOV    A,R2      ;ROW SCAN SELECT TO ACCUMULATOR
005C 4320     224      ORL    A,#20H    ;SAVE INTERRUPT, NO CHANGE TO SCAN VALUE
005E AA      225      MOV    R2,A      ;ROW SCAN SELECT REGISTER
005F 232F     226 ADJFIN: MOV    A,#FIFOTA  ;FIFO TOP ADDRESS TO ACCUMULATOR
0061 DC      227      XRL    A,R4      ;COMPARE WITH CURRENT FIFO INPUT ADDRESS
0062 C667     228      JZ     RSFFIN    ;IF THE SAME RESET FIFO INPUT REGISTER
0064 1C      229      INC     R4      ;NEXT FIFO INPUT ADDRESS
0065 0469     230      JMP     CHFFUL    ;CHECK FIFO FULL
0067 BC08     231 RSFFIN: MOV    R4,#FIFOBA ;RESET FIFO INPUT REGISTER,BOTTOM OF FIFO
0069 FC      232 CHFFUL: MOV    A,R4      ;FIFO INPUT ADDRESS TO ACCUMULATOR
006A DD      233      XRL    A,R5      ;COMPARE INPUT WITH OUTPUT FIFO ADDRESS
006B 967D     234      JNZ     CHCNTR    ;IF NOT SAME CHECK COLUMN COUNTER VALUE
006D 866D     235 CHOBFI: J0BF    CHOBFI    ;IF OBF IS 1 THEN CHECK OBF
006F 232F     236 ADJFOT: MOV    A,#FIFOTA  ;FIFO TOP ADDRESS TO ACCUMULATOR
0071 DD      237      XRL    A,R5      ;COMPARE TOP TO OUTPUT FIFO ADDRESS
0072 C677     238      JZ     RSFFOT    ;IF THE SAME RESET FIFO OUTPUT REGISTER
0074 1D      239      INC     R5      ;NEXT FIFO OUTPUT ADDRESS
0075 0479     240      JMP     LOADDB    ;LOAD DBBOUT
0077 BD08     241 RSFFOT: MOV    R5,#FIFOBA ;RESET FIFO OUTPUT ADDRESS TO BOTTOM OF FIFO
0079 FD      242 LOADDB: MOV    A,R5      ;OUTPUT FIFO ADDRESS TO ACCUMULATOR
007A A9      243      MOV    R1,A      ;FIFO POINTER USED FOR OUTPUT
007B F1      244      MOV    A,@R1    ;CHANGE WORD TO ACCUMULATOR
007C 02      245      OUT     DBB,A      ;CHANGE WORD TO DBBOUT
007D FB      246 CHCNTR: MOV    A,R3      ;COLUMN COUNTER TO ACCUMULATOR
007E 963A     247      JNZ     RRLOOK    ;IF NOT 0 FINISH CHANGE WORD ENCODING
0080 2308     248 CHFFEM: MOV    A,#FIFOBA ;FIFO BOTTOM ADDRESS TO ACCUMULATOR
0082 DC      249      XRL    A,R4      ;COMPARE FIFO INPUT ADDRESS WITH FIFO BOTTOM ADDRESS
0083 C68C     250      JZ     ADJFEM    ;IF THE SAME,ADJUST TO CHECK FOR FIFO EMPTY
0085 FC      251      MOV    A,R4      ;FIFO INPUT ADDRESS TO ACCUMULATOR
0086 07      252      DEC     A      ;DECREMENT FIFO INPUT ADDRESS IN ACCUMULATOR
0087 DD      253      XRL    A,R5      ;COMPARE INPUT TO OUTPUT FIFO ADDRESSES

```

APPENDIX B1 (Continued)

LOC	OBJ	SEQ	SOURCE STATEMENT	
0088	C691	254	JZ STATMT	; IF SAME, WRITE STATUS REGISTER FOR FIFO EMPTY
008A	049C	255	JMP CH0BF2	; CHECK 0BF
008C	232F	256	ADJFEM: MOV A, #FIF0A	; FIFO TOP ADDRESS TO ACCUMULATOR
008E	DD	257	XRL A, R5	; COMPARE TOP TO OUTPUT FIFO ADDRESS
008F	969C	258	JNZ CH0BF2	; IF NOT SAME THEN FIFO IS NOT EMPTY, CHECK 0BF
0091	2300	259	STATMT: MOV A, #00H	; CLEAR BIT 0 FOR FIFO EMPTY
0093	90	260	MOV STS, A	; WRITE TO STATUS REGISTER
0094	9ADF	261	INTRLO: ANL P2, #0DFH	; FIFO EMPTY, INTERRUPT PORT 2-5 LOW
0096	FA	262	MOV A, R2	; SCAN ROW SELECT TO ACCUMULATOR
0097	53DF	263	ANL A, #0DFH	; SAVE INTERRUPT, NO CHANGE TO SCAN VALUE
0099	AA	264	MOV R2, A	; SCAN ROW SELECT REGISTER
009A	041D	265	JMP ADJREG	; ADJUST REGISTERS
009C	861D	266	CH0BF2: J0BF ADJREG	; IF 0BF=1 THEN ADJUST REGISTERS
009E	046F	267	JMP ADJFOT	; ADJUST FIFO OUT ADDRESS TO LOAD 0BB0UT
		268	;	
		269	END	

USER SYMBOLS

ADJFEM 008C	ADJFIN 005F	ADJFOT 006F	ADJREG 001D	CHCNTR 007D	CHFFEM 0080	CHFFUL 0069	CH0BF1 006D
CH0BF2 009C	DELAY2 0030	ENCODE 0045	FIF0BA 0008	FIF0TA 002F	FILLMX 000D	INITMX 0000	INTRH1 0059
INTRLO 0094	LOADDB 0079	LOADFF 0052	0BFINT 0018	RRLOOK 003A	RSETRG 0026	RSFFIN 0067	RSFFOT 0077
SCANMX 002C	SCANTM 000F	STATMT 0091	STATNE 0056				

ASSEMBLY COMPLETE, NO ERRORS

APPENDIX B2

ISIS-II 8080/8085 MACRO ASSEMBLER, X108

MODULE PAGE 1

8085A/UI SENSOR MATRIX CONTROLLER

LOC	OBJ	SEQ	SOURCE STATEMENT
		1 ;	
		2 ;	SUBROUTINE TO READ ALL CHANGES IN THE UPI AND BUILD A BUFFER
		3 ;	STARTING AT BUFSRT. REG. B CONTAINS THE NUMBER OF CHANGES
		4 ;	UPON EXIT. THE MAXIMUM NUMBER OF CHANGES IN ANY ONE CALL
		5 ;	IS 255.
		6 ;	
		7 ;	INPUTS: NOTHING
		8 ;	OUTPUTS: CHANGE WORD BUFFER AT BUFSRT
		9 ;	CHANGE WORD COUNT IN REG. B
		10 ;	CALLS: NOTHING
		11 ;	
4000		12	ORG 4000H
00E5		13	STATUS EQU 0E5H ;UPI STATUS PORT
00E4		14	DBBOUT EQU 0E4H ;UPI DBBOUT PORT
0010		15	FIFO EQU 10H ;FIFO NOT EMPTY MASK
0001		16	OBF EQU 01H ;OBF MASK
4300		17	BUFSRT EQU 4300H ;BUFFER START LOCATION
		18 ;	
4000 210043		19	START: LXI H, BUFSRT ;INITIALIZE BUFFER POINTER
4003 0600		20	MVI B, 00H ;CLEAR CHANGE WORD COUNTER
4005 DBE5		21	POLL1: IN STATUS ;READ UPI STATUS
4007 E611		22	ANI FIFO OR OBF ;TEST FIFO NOT EMPTY AND OBF
4009 C8		23	RZ ;RETURN IF ZERO
400A DBE5		24	IN STATUS ;READ UPI STATUS
400C E601		25	ANI OBF ;TEST OBF FLAG
400E CA0540		26	JZ POLL1 ;WAIT IF NOT READY
4011 DBE4		27	IN DBBOUT ;READ CHANGE WORD
4013 77		28	MOV M, A ;LOAD BUFFER WITH CHANGE WORD
4014 23		29	INX H ;INC BUFFER POINTER
4015 04		30	INR B ;INC CHANGE WORD COUNTER
4016 C8		31	RZ ;EXIT IF COUNTER = 256
4017 C30540		32	JMP POLL1 ;CHECK IF MORE CHANGE WORDS
		33 ;	
		34	END

2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100

101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200

Appendix C1

APPENDIX C1

ISIS-II MCS-48/UPI-41 MACRO ASSEMBLER, V2.0
AP-41 COMBINATION I/O DEVICE

LOC	OBJ	SEQ	SOURCE STATEMENT
1			\$MOD42
2			*****UNIOD*****
3			;
4			; THIS UPI-41 PROGRAM IMPLEMENTS A FULL-DUPLEX UART WITH ON-CHIP
5			; BAUD RATE GENERATION IN COMBINATION WITH AN 8-BIT PARALLEL I/O
6			; PORT. THE BAUD RATE IS SELECTABLE FROM 110 TO 1200 BAUD. THE
7			; PARALLEL I/O PORT IS PROGRAMMABLE FOR EITHER INPUT OR OUTPUT.
8			;
9			; INTERRUPT OUTPUTS ARE AVAILABLE FOR DATA AVAILABLE ON THE RECEIVER
10			; AND PARALLEL INPUT. THE STATUS REGISTER MUST BE READ TO DETERMINE
11			; WHICH SOURCE CAUSED THE INTERRUPT. THE FLAGS F0 AND F1 CODE THE
12			; INTERRUPT SOURCE. F0 AND F1 ALSO GIVE AN INDICATION OF COMMAND
13			; ERRORS.
14			;
15			*****
16			;
17			; REGISTER DEFINITION
18			RB0 RB1
19			---
20			0 NOT USED NOT USED
21			1 NOT USED BAUD RATE CONSTANT
22			2 NOT USED TX TICK COUNTER
23			3 RX STATUS (RXSTS) TX SERIALIZER
24			4 RX HOLDING TX BUFFER
25			5 RX TICK COUNTER TX STATUS (TXSTS)
26			6 RX DESERIALIZER COMMAND STORE
27			7 STATUS REG STORE ACC. INTERRUPT SAVE
28			;
29			*****
30			;
31			\$EJECT

APPENDIX C1 (Continued)

LOC	OBJ	SEQ	SOURCE STATEMENT
32			;
33			*****
34			;
35			COMMANDS
36			;
37			CONFIGURE: 0 0 0 A B C D P
38			A - 1200 BAUD SELECT
39			B - 600 BAUD SELECT
40			C - 300 BAUD SELECT
41			D - 110 BAUD SELECT
42			E - PARALLEL I/O DIRECTION
43			0 - INPUT
44			1 - OUTPUT
45			;
46			I/O: 1 0 0 0 0 0 0 0 (PERFORM I/O OPERATION)
47			RESET ERROR: 1 1 0 0 0 0 0 0 (RESET RX ERROR IN STATUS)
48			;
49			*****
50			;
51			STATUS REGISTER DEFINITION
52			;
53			BIT DEFINITION
54			---
55			0 OBF - DATA AVAILABLE
56			1 IBF - BUSY
57			2 F0
58			3 F1
59			4 NOT USED
60			5 TXINT - TX INTERRUPT
61			6 FRAMING ERROR
62			7 OVERRUN ERROR
63			;
64			F0 F1 OPERATION
65			---
66			0 0 X
67			0 1 PARALLEL I/O DATA AVAILABLE
68			1 0 SERIAL I/O DATA AVAILABLE
69			1 1 COMMAND ERROR
70			;
71			*****
72			;
73			\$EJECT

APPENDIX C1 (Continued)

LOC	OBJ	SEQ	SOURCE STATEMENT
74			;
75			*****
76			;
77			STATUS REGISTER DEFINITIONS
78			;
79			RXSTS
80			TXSTS
81			0 RX FLAG - SPACE TX FLAG - TRANSMITTING CHR
82			1 START FLAG - GOOD START REQUEST BYTE - CHR IN BUFFER
83			2 BYTE FINISHED TX PIPELINED DATA BIT
84			3 DATA READY START BIT FLAG
85			4 FRAMING ERROR NOT USED
86			5 OVERRUN ERROR NOT USED
87			6 IO DIRECTION NOT USED
88			7 IO FLAG NOT USED
89			;
90			*****
91			;
92			PORT 2 DEFINITIONS
93			;
94			BIT DEFINITION
95			---
96			0 TX DATA
97			1 NOT USED
98			2 NOT USED
99			3 TX INTERRUPT
100			4 OBF INTERRUPT (RX OR I/O DATA AVAILABLE)
101			5 NOT USED
102			6 NOT USED (TICK SAMPLE)
103			7 NOT USED
104			;
105			*****
106			;
107			MISC.
108			;
109			RX DATA T0 INPUT
110			EXT CLOCK T1 INPUT 76.8KHZ (1.2288MHZ/16)
111			;
112			*****
113			;
114			\$EJECT

APPENDIX C1 (Continued)

```

117 ;
118 ;SYSTEM EQUATES:
119 ;
0001 120 RXFLG EQU 01H ;RECEIVE FLAG IN RXSTS
0002 121 SRIFLG EQU 02H ;START BIT FLAG IN RXSTS
0004 122 BFFLG EQU 04H ;BYTE FINISHED FLAG IN RXSTS
0008 123 DATRDY EQU 08H ;DATA READY FLAG IN RXSIS
0010 124 FRAMER EQU 10H ;FRAMING ERROR FLAG IN RXSTS
0020 125 OVRUN EQU 20H ;OVERRUN ERROR FLAG IN RXSTS
0040 126 IODIR EQU 40H ;I/O DIRECTION FLAG IN RXSIS
0080 127 IOFLG EQU 80H ;I/O REQUEST FLAG IN RXSTS
0001 128 TXFLG EQU 01H ;TX FLAG IN TXSTS
0002 129 REQFLG EQU 02H ;REQUEST BYTE FLAG IN TXSTS
0040 130 TICOUT EQU 40H ;TICK SAMPLE BIT IN PORT 2
0080 131 RXINTL EQU 80H ;RX DESERIALIZER INITIALIZATION
0004 132 TICSRT EQU 04H ;TICK INITIALIZATION
007F 133 ASCMSK EQU 7FH ;ASCII MASK
0003 134 TXTIC EQU 03H ;TX TICK MOD MASK
0028 135 TXEND EQU 40D ;TICK COUNT AT END OF TX CHARACTER
0024 136 STPEND EQU 36D ;TICK COUNT AT END OF TX DATA
0004 137 MARK EQU 04H ;MARK OUTPUT
00FB 138 SPACE EQU 0FBH ;SPACE OUTPUT
0000 139 ZERO EQU 00H ;GENERAL CLEAR
0008 140 TXINT EQU 08H ;TX INTERRUPT OUTPUT IN PORT 2
0020 141 TXBIT EQU 20H ;TX INTERRUPT BIT IN STATUS
0020 142 TIMCON EQU 32D ;TIMER CONSTANT RAM LOCATION
003F 143 RSTERR EQU 3FH ;RESET ERROR MASK FOR STATUS
0040 144 FSTS EQU 40H ;FRAMING ERROR BIT IN STATUS
0080 145 OVSTS EQU 80H ;OVERRUN ERROR BIT IN STATUS
0001 146 MKOUT EQU 01H ;MARK OUTPUT TO PORT
00FE 147 SPOUT EQU 0FEH ;SPACE OUTPUT TO PORT
0008 148 SBIT EQU 08H ;TX START BIT FLAG
0003 149 RXSTS EQU R3 ;RX STATUS REGISTER
0005 150 TXSTS EQU R5 ;TX STATUS REGISTER
151 ;
152 $EJECT

```


APPENDIX C1 (Continued)

LOC	OBJ	SEQ	SOURCE STATEMENT
		153	*****
		154	;
		155	;RESET VECTOR LOCATION
		156	;
		157	*****
		158	;
0000		159	ORG 0000H
		160	;
0000 C5		161	RESET: SEL R00 ;GET INTO R00 AT RESET
0001 4400		162	JMP INIT ;GO TO INITIALIZATION
		163	;
		164	*****
		165	;
		166	;TIMER INTERRUPT LOCATION - TIMER IS SET TO 4 TIMES THE BAUD RATE. THE
		167	;RECEIVER AND TRANSMITTER ARE SERVICED EVERY FOUR TIMER TICKS. SOFTWARE
		168	;DELAY LOOP IS USED FOR TIMING FINE-TUNING. R01 R1 POINTS AT DELAY
		169	;CONSTANT AT INTERRUPT. R1-1 POINTS AT TIMER CONSTANT.
		170	;
		171	*****
		172	;
0007		173	ORG 0007H
		174	;
0007 D5		175	TIMINT: SEL R01 ;INTERRUPT PROCESSING IN R01
0008 AF		176	MOV R7,A ;SAVE ACCUMULATOR IN R7
0009 F9		177	MOV A,R1 ;GET TIMER CONSTANT
000A 00		178	NOP ;DELAY TO GET INTO T1 HIGH
000B 560B		179	INT1: JT1 INT1 ;WAIT UNTIL T1 IS LOW
000D 62		180	MOV T,A ;THEN LOAD COUNTER
		181	;
		182	;TICK SAMPLE OUTPUT
		183	;
000E 9ABF		184	ANL P2,#NOT TICOUT
0010 8A40		185	ORL P2,#TICOUT
		186	;
		187	*****
		188	;
		189	;TRANSMITTER OUTPUT - TIME CRITICAL TASKS DONE FIRST. DATA BIT OUTPUT
		190	;PIPELINED IN TXSTS BIT 2 IS OUTPUT NOW.
		191	;
		192	*****
		193	;
0012 FD		194	TXOUT: MOV A,TXSTS ;GET TX STATUS
0013 5219		195	JB2 MOUT ;TEST PIPELINED DATA
0015 9AFE		196	ANL P2,#SPOUT ;OUTPUT SPACE IF RESET
0017 041B		197	JMP RCV ;DO RECEIVER
0019 8A01		198	MOUT: ORL P2,#MKOUT ;OUTPUT MARK IS SET
		199	;
		200	*****
		201	;
		202	;START OF RECEIVER FLOW - RXSTS REGISTER
		203	;HOLDS RECEIVER STATUS.
		204	;
		205	*****
		206	;
001B C5		207	RCV: SEL R00 ;SWITCH TO RX BANK

APPENDIX C1 (Continued)

LOC	OBJ	SEQ	SOURCE STATEMENT
001C	FB	208	MOV A, RXSTS ; GET RXSTS
001D	1226	209	JB0 RCV1 ; TEST RECEIVE FLAG
		210	; 0 - NO CHR BEING RECEIVED
		211	; 1 - POSSIBLE START BIT, DO TEST
001F	3668	212	JT0 XMIT ; TEST RXD INPUT
		213	; 0 - SPACE, SET RX FLAG
		214	; 1 - MARK, GO CHECK XMIT
0021	4301	215	ORL A, #RXFLG ; SPACE - SET RX FLAG
0023	AB	216	MOV RXSTS, A ; RESTORE RXSTS
0024	0468	217	JMP XMIT ; GO HANDLE XMTR
		218	;
		219	; START BIT TEST
		220	;
0026	3238	221	RCV1: JB1 RCV3 ; FIRST TEST START BIT FLAG
0028	3633	222	JT0 RCV2 ; TEST RXD INPUT
		223	; 0 - SPACE, GOOD START BIT
		224	; 1 - MARK, BAD START BIT, IGNORE
002H	4302	225	ORL A, #SRTFLG ; GOOD START - SET START BIT FLAG
002C	AB	226	MOV RXSTS, A ; RESTORE RXSTS
002D	BE80	227	MOV R6, #RXINTL ; SETUP RX DESERIALIZER
002F	BD04	228	MOV R5, #TICSRT ; LOAD RX TICK COUNTER
0031	0468	229	JMP XMIT ; GO HANDLE XMTR
		230	;
		231	; BAD START BIT - RESET FLAGS
		232	;
0033	53FE	233	RCV2: ANL A, #NOT RXFLG ; RESET RECEIVE FLAG
0035	AB	234	MOV RXSTS, A ; RESTORE RXSTS
0036	0468	235	JMP XMIT ; GO HANDLE XMTR
		236	;
		237	; IN MIDDLE OF CHR - SAMPLE EVERY 4 TIMER TICKS
		238	;
0038	ED68	239	RCV3: DJNZ R5, XMIT ; WAIT UNTIL 4TH TICK
003A	BD04	240	MOV R5, #TICSRT ; RELOAD RX TICK COUNTER
003C	524D	241	JB2 RCV5 ; TEST BYTE FINISHED FLAG
		242	; 0 - MIDDLE OF CHR, CONTINUE
		243	; 1 - DONE WITH STOP BITS
003E	97	244	CLR C ; CLEAR CARRY BEFORE ROTATE
003F	2642	245	JNT0 RCV4 ; TEST RXD INPUT
0041	A7	246	CPL C ; RXD IS MARK, SET CARRY
0042	FE	247	RCV4: MOV A, R6 ; GET DESERIALIZER
0043	67	248	RRC A ; ROTATE IN NEW BIT
0044	AE	249	MOV R6, A ; RESTORE DESERIALIZER
0045	E668	250	JNC XMIT ; TEST CARRY AFTER ROTATE
		251	; 0 - MIDDLE OF CHR
		252	; 1 - STOP BIT COMING NEXT
0047	FB	253	MOV A, RXSTS ; GET RXSTS
0048	4304	254	ORL A, #BFFLG ; SET BYTE FINISHED FLAG
004A	AB	255	MOV RXSTS, A ; RESTORE RXSTS
004B	0468	256	JMP XMIT ; GO HANDLE XMTR
		257	;
		258	; BYTE FINISHED - DO STOP BIT TEST
		259	;
004D	2660	260	RCV5: JNT0 RCV8 ; TEST RXD INPUT
		261	; 0 - SPACE, INVALID STOP BIT
		262	; 1 - MARK, VALID STOP BIT

APPENDIX C1 (Continued)

LOC	OBJ	SEQ	SOURCE STATEMENT
004F	53EF	263	ANL A,#NOT FRAMER ; NO FRAMING ERROR, RESET FLAG
		264	;
		265	; OVERRUN TEST - IF RX DATA READY STILL SET, OVERRUN ERROR
		266	;
0051	7264	267	RCV6: JB3 RCV9 ; IF DATA READY STILL SET, ERROR
0053	53DF	268	ANL A,#NOT OVRUN ; NO OVERRUN, RESET FLAG
		269	;
		270	; CLEAN UP RXSTS AT CHR COMPLETE
		271	;
0055	4308	272	RCV7: ORL A,#DATRDY ; SET DATA READY
0057	53F8	273	ANL A,#NOT (RXFLG OR SRTFLG OR BFFLG) ; RESET OTHER FLAGS
0059	AB	274	MOV RXSTS,A ; RESTORE RXSTS
005A	FE	275	MOV A,R6 ; GET DESERIALIZER REG
005B	537F	276	ANL A,#ASCHSK ; MAKE IT 7 BITS
005D	AC	277	MOV R4,A ; PUT DATA INTO HOLDING REG
005E	0468	278	JMP XMIT ; GO HANDLE XMITR
		279	;
		280	; BAD STOP - SET FRAMING ERROR FLAG
		281	;
0060	4310	282	RCV8: ORL A,#FRAMER ; SET FRAMING ERROR FLAG
0062	0451	283	JMP RCV6 ; CONTINUE
		284	;
		285	; OVERRUN ERROR - SET OVERRUN FLAG
		286	;
0064	4320	287	RCV9: ORL A,#OVRUN ; SET OVERRUN FLAG
0066	0455	288	JMP RCV7 ; CONTINUE
		289	;
		290	*****
		291	;
		292	; START OF TRANSMITTER FLOW - TRANSMITTER IS SERVICED EVERY 4 TICKS.
		293	; THE TX TICK COUNTER SERVES AS THE TX BIT COUNTER. TRANSMITTER STATUS
		294	; IS HELD IN THE TXSTS REGISTER.
		295	;
		296	*****
		297	;
0068	D5	298	XMIT: SEL RB1 ; BE SURE WE'RE IN RB1
0069	FD	299	MOV A,TXSTS ; GET TX STATUS
006A	72B3	300	JB3 SRTBIT ; THIS IS START OF START BIT
006C	1A	301	INC R2 ; INC TX TICK COUNTER
006D	2303	302	MOV A,#1XTIC ; TEST TICK COUNTER MOD 4
006F	5A	303	ANL A,R2
0070	96B0	304	JNZ RETURN ; IF NON-ZERO, MIDDLE OF BIT
0072	FD	305	MOV A,TXSTS ; ZERO, GET TXSTS
0073	37	306	CPL A ; COMPLEMENT FOR 0 TEST
0074	129C	307	JB0 XMT4 ; TEST TX FLAG
		308	; 0 - NOT TX'ING, CHECK FOR NEW CHR
		309	; 1 - CURRENTLY IN CHR
0076	2328	310	MOV A,#TXEND ; CHECK FOR END OF DATA AND STOP
0078	DA	311	XRL A,R2 ; XOR WITH CURRENT TICK COUNT
0079	96B1	312	JNZ XMT1 ; NOT DONE, CONTINUE
007B	FD	313	MOV A,TXSTS ; DONE, GET TXSTS
007C	53FE	314	ANL A,#NOT TXFLG ; RESET TX FLAG
007E	AD	315	MOV TXSTS,A ; RESTORE TXSTS
007F	04B0	316	JMP RETURN ; GO EXIT
		317	;

APPENDIX C1 (Continued)

```

0081 2324 320 XMT1: MOV A, #STPEND ;CHECK FOR STOP BIT TIME
0083 DA 321 XRL A, R2 ;COMPARE WITH TICK COUNTER
0084 968C 322 JNZ XMT2 ;NOT TIME, DO NEXT BIT
323 ;
324 ;TRANSMIT STOP BIT
325 ;
0086 FD 326 MOV A, TXSTS ;GET TX STATUS
0087 4384 327 ORL A, #MARK ;SETUP PIPELINED STOP BIT
0089 AD 328 MOV TXSTS, A ;RESTORE TX STATUS
008A 04B0 329 JMP RETURN ;RETURN
330 ;
331 ;IN MIDDLE OF CHR - TRANSMIT NEXT BIT
332 ;
008C FB 333 XMT2: MOV A, R3 ;GET TX SERIALIZER
008D 67 334 RRC A ;ROTATE NEXT BIT INTO CARRY
008E AB 335 MOV R3, A ;RESTORE SERIALIZER
008F FD 336 MOV A, TXSTS ;GET TX STATUS FOR PIPELINED DATA
0090 F697 337 JC XMT3 ;OUTPUT A MARK IF 1
0092 53FB 338 ANL A, #SPACE ;RESET TXDATA BIT
0094 AD 339 MOV TXSTS, A ;RESTORE TX STATUS
0095 04B0 340 JMP RETURN ;GO EXIT
0097 4384 341 XMT3: ORL A, #MARK ;SET TXDATA BIT
0099 AD 342 MOV TXSTS, A ;RESTORE TX STATUS
009A 04B0 343 JMP RETURN ;GO EXIT
344 ;
345 ;TEST REQUEST FLAG SINCE NOT CURRENTLY TRANSMITTING
346 ;
009C 32A8 347 XMT4: JB1 XMT5 ;TEST TX REQUEST FLAG
348 ;0 - NO CHR WAITING IN BUFFER
349 ;1 - CHR WAITING IN BUFFER
009E FC 350 MOV A, R4 ;CHR WAITING, GET IT FROM HOLDING
009F AB 351 MOV R3, A ;PUT IN SERIALIZER
00A0 FD 352 MOV A, TXSTS ;GET TXSTS
00A1 53FD 353 ANL A, #NOT REQFLG ;RESET REQUEST FLAG
00A3 4309 354 ORL A, #TXFLG OR SBIT ;SET TX AND START BIT FLAGS
00A5 53FB 355 ANL A, #SPACE ;SETUP TXDATA FOR START BIT
00A7 AD 356 MOV TXSTS, A ;RESTORE TXSTS
357 ;
358 ;TX BUFFER EMPTY - SET TXINT PIN AND BIT
359 ;
00A8 8A08 360 XMT5: ORL P2, #TXINT ;SET TXINT PIN
00AA C5 361 SEL RB0 ;SWITCH FOR S1S
00AB FF 362 MOV A, R7 ;GET STS
00AC 4320 363 ORL A, #TXBIT ;SET TXINT BIT
00AE AF 364 MOV R7, A ;RESTORE STS
00AF 90 365 MOV STS, A ;LOAD STATUS
366 ;
367 ;*****
368 ;
369 ;EXIT FOR TIMER INTERRUPT ROUTINE POINT
370 ;
371 ;*****
372 ;

```

APPENDIX C1 (Continued)

LOC	OBJ	SEQ	SOURCE STATEMENT	INSTR	OP	VAL	DISP
00B0	D5	373	RETURN: SEL RB1 ;MAKE SURE WE'RE IN RB1	SEL	RB1		
00B1	FF	374	MOV A,R7 ;RESTORE A	MOV	A,R7		
00B2	93	375	RETR ;RETURN WITH RESTORE	RETR			
		376	;				
		377	*****				
		378	;				
		379	;GET HERE IF INTERRUPT IS FIRST FOR START BIT - CLEAR START BIT FLAG IN				
		380	TXSTS AND SETUP TX TICK COUNTER.				
		381	;				
		382	*****				
		383	;				
00B3	53F7	384	SRTBIT: ANL A,#NOT SBIT ;RESET START BIT FLAG IN TXSTS	ANL	A,#NOT SBIT		
00B5	AD	385	MOV TXSTS,A ;RESTORE TX STATUS	MOV	TXSTS,A		
00B6	BA01	386	MOV R2,#01H ;INITIALIZE TX TICK COUNTER	MOV	R2,#01H		
00B8	04B0	387	JMP RETURN ;RETURN	JMP	RETURN		
		388	;				
		389	\$EJECT				

APPENDIX C1 (Continued)

LOC	OBJ	SEQ	SOURCE STATEMENT
		390 ;	
		391 ;*****	
		392 ;	
		393 ;COMMAND RECOGNIZER - GET HERE FROM IBF WRITE WITH F1 SET. COMMAND	
		394 ;IS STORED IN R6. BAUD RATE SELECTION BITS ARE EVALUATED RIGHT TO LEFT.	
		395 ;THE FIRST SET BIT FOUND DETERMINES THE BAUD RATE. IF AN INVALID COMMAND	
		396 ;IS DETECTED, BOTH F1 AND F0 ARE SET AND NO ACTION IS TAKEN.	
		397 ;THE TIMER BAUD RATE CONSTANT IS SET TO TWO COUNTS LESS THAN THE DESIRED	
		398 ;NUMBER.	
		399 ;	
		400 ;*****	
0100		401 ;	
		402 ORG 0100H	
		403 ;	
0100 D5		404 CMD: SEL RB1	;SELECT RB1
0101 22		405 IN A,DBB	;READ COMMAND
0102 AE		406 MOV R6,A	;SAVE COMMAND IN R6
0103 F227		407 JB7 IOER	;IF BIT 7 SET, IO OPERATION
0105 53E0		408 ANL A,#0E0H	;TEST TOP 3 BITS
0107 963A		409 JNZ ERROR	;IF NON-ZERO, ERROR
0109 C5		410 SEL RB0	;IO FLAG IN RB0
010A 1221		411 JB0 CMD2	;IF BIT 0=1, OUTPUT PORT
010C 89FF		412 ORL P1,#0FFH	;INPUT PORT, SET ALL HIGH
010E FB		413 MOV A,RXSTS	;GET RXSTS
010F 53BF		414 ANL A,#NOT IODIR	;RESET IO DIRECTION FLAG
0111 AB		415 MOV RXSTS,A	;RESTORE RXSTS
0112 D5		416 CMD1: SEL RB1	;BAUD RATE CONSTANTS IN RB1
0113 B920		417 MOV R1,#TIMCON	;POINT AT TIMER CONSTANT LOCATION
0115 FE		418 MOV A,R6	;GET COMMAND
0116 323E		419 JB1 B110	;110 BAUD SELECTED
0118 5242		420 JB2 B300	;300 BAUD SELECTED
011A 7246		421 JB3 B600	;600 BAUD SELECTED
011C 924A		422 JB4 B1200	;1200 BAUD SELECTED
011E B5		423 CPL F1	;RESET F1
011F 4414		424 JMP MNLP1	;DONE, JUMP BACK TO MAIN LOOP
		425 ;	
		426 ;PORT IS SELECTED AS OUTPUT PORT - SET IO DIRECTION FLAG	
		427 ;	
0121 FB		428 CMD2: MOV A,RXSTS	;GET RXSTS
0122 4340		429 ORL A,#IODIR	;SET IO DIRECTION FLAG
0124 AB		430 MOV RXSTS,A	;RESTORE RXSTS
0125 2412		431 JMP CMD1	;CONTINUE
		432 ;	
		433 ;HERE WITH EITHER IO OR RESET ERROR COMMAND	
		434 ;	
0127 D231		435 IOER: JB6 ERRST	;IF BIT 6 SET, RESET ERROR FLAGS
0129 C5		436 SEL RB0	;IO FLAG IN RB0
012A FB		437 MOV A,RXSTS	;GET RXSTS
012B 4380		438 ORL A,#IOFLG	;SET IO FLAG
012D AB		439 MOV RXSTS,A	;RESTORE RXSTS
012E B5		440 CPL F1	;RESET F1
012F 4414		441 JMP MNLP1	;DONE, JUMP BACK TO MAIN LOOP
		442 ;	
		443 ;RESET ERROR COMMAND	
		444 ;	

APPENDIX C1 (Continued)

LOC	OBJ	SEQ	SOURCE STATEMENT
0131	C5	445	EKRST: SEL R00 ; STS IN R00
0132	FF	446	MOV A, R7 ; GET STS
0133	533F	447	ANL A, #RSTERR ; RESET ERROR FLAGS
0135	AF	448	MOV R7, A ; RESTORE STS
0136	90	449	MOV STS, A ; LOAD STATUS
0137	B5	450	CPL F1 ; RESET F1
0138	4414	451	JMP MNLP1 ; DONE, BACK TO MAIN LOOP
		452 ;	
		453	; COMMAND ERROR - SET BOTH F1 AND F0
		454 ;	
013A	85	455	ERROR: CLR F0 ; SET F0
013B	95	456	CPL F0
013C	4414	457	JMP MNLP1 ; DONE, BACK TO MAIN LOOP
		458 ;	
		459	; 110 BAUD CONSTANTS
		460 ;	
013E	B954	461	B110: MOV R1, #- (1740-2D) ; LOAD 110 BAUD CONSTANT
0140	244C	462	JMP STTIMR ; GO START TIMER
		463 ;	
		464	; 300 BAUD CONSTANTS
		465 ;	
0142	B9C2	466	B300: MOV R1, #- (640-2D) ; LOAD 300 BAUD CONSTANT
0144	244C	467	JMP SITIMR ; GO START COUNTER
		468 ;	
		469	; 600 BAUD CONSTANTS
		470 ;	
0146	B9E2	471	B600: MOV R1, #- (320-2D) ; LOAD 600 BAUD CONSTANT
0148	244C	472	JMP STTIMR ; GO START COUNTER
		473 ;	
		474	; 1200 BAUD CONSTANTS
		475 ;	
014A	B9F2	476	B1200: MOV R1, #- (160-2D) ; LOAD 1200 BAUD CONSTANT
		477 ;	
		478	; START COUNTER
		479 ;	
014C	F9	480	STTIMR: MOV A, R1 ; GET COUNTER CONSTANT
014D	62	481	MOV T, A ; LOAD COUNTER
014E	45	482	STRT CNT ; START COUNTER
014F	25	483	EN TCNTI ; ENABLE TIMER INTERRUPTS
0150	B5	484	CPL F1 ; RESET F1
0151	4414	485	JMP MNLP1 ; DONE, BACK TO MAIN LOOP
		486 ;	
		487	; EJECT

```

489 ;*****
490 ;
491 ;DATA ROUTINE - GET HERE WITH I/O WRITE WITH F1 RESET. THIS ROUTINE
492 ;FIRST TESTS IF THE I/O FLAG IS SET IN THE RXST5 REGISTER. IF SO, THE DATA
493 ;IS FOR THE OUTPUT PORT. OTHERWISE, THE DATA IS FOR THE TRANSMITTER AND
494 ;IS PLACED IN THE TX BUFFER REGISTER. THE TXINT BIT AND PIN ARE RESET.
495 ;
496 ;*****
497 ;
0153 C5 498 DATA: SEL R80 ;DATA HANDLED MOSTLY IN R80
0154 FB 499 MOV A,RXST5 ;GET RXST5
0155 F267 500 JB7 IODATA ;IF I/O FLAG SET, DATA IN FOR I/O
0157 FF 501 MOV A,R7 ;GET STS
0158 53DF 502 ANL A,#NOT TXBIT ;RESET TXINT BIT IN STS
015A AF 503 MOV R7,A ;RESTORE STS
015B 90 504 MOV STS,A ;LOAD STATUS
015C 9AF7 505 ANL P2,#NOT TXINT ;RESET TXINT PIN
015E D5 506 SEL RB1 ;TXST5 IN RB1
015F 22 507 IN A,DBB ;READ DATA
0160 AC 508 MOV R4,A ;PUT DATA IN TX BUFFER
0161 FD 509 MOV A,TXST5 ;GET 1XST5
0162 4302 510 ORL A,#REQFLG ;SET REQUEST FLAG IN TXST5
0164 AD 511 MOV TXST5,A ;RESTORE 1XST5
0165 4414 512 JMP MNLP1 ;BACK TO MAIN LOOP
513 ;
514 ;IO DATA ROUTINE
515 ;
0167 537F 516 IODATA: ANL A,#NOT IOFLG ;RESET IO FLAG
0169 AB 517 MOV RXST5,A ;RESTORE RXST5
016A 22 518 IN A,DBB ;READ IO DATA FROM DBBIN
016B 39 519 OUTL P1,A ;OUTPUT TO PORT 1
016C 4414 520 JMP MNLP1 ;DONE, BACK TO MAIN LOOP
521 ;
522 $EJECT

```

APPENDIX C1 (Continued)

LOC	OBJ	SEQ	SOURCE STATEMENT
		523 ;	
		524 ;*****	
		525 ;	
		526 ;INITIALIZATION - GET HERE AT RESET. THIS ROUTINE RESETS THE INTERRUPT	
		527 ;OUTPUTS AND ENABLES THEM, AND CLEARS THE APPROPRIATE STATUS AND DATA	
		528 ;REGISTERS.	
		529 ;	
		530 ;*****	
		531 ;	
0200		532 ORG 0200H	
		533 ;	
0200 9AF7		534 INIT: ANL P2,#0F7H ;RESET TXIN1 PIN	
0202 F5		535 EN FLAGS ;ENABLE INTERRUPTS OUTPUT	
0203 2300		536 MOV A,#ZERO ;CLEAR A	
0205 AB		537 MOV RXSTS,A ;CLEAR RXSTS	
0206 AD		538 MOV R5,A ;CLEAR RX TICK COUNTER	
0207 AF		539 MOV R7,A ;CLEAR STS	
0208 D5		540 SEL RB1 ;SWITCH BANKS	
0209 AE		541 MOV R6,A ;CLEAR CONFIGURE STORE	
020A BD04		542 MOV TXSTS,#MARK ;SETUP PIPELINED TX DATA	
		543 ;	
		544 ;*****	
		545 ;	
		546 ;MAIN LOOP - IBF AND OBF ARE HANDLED IN THIS LOOP. IF IBF=1, THE	
		547 ;APPROPRIATE COMMAND OR DATA ROUTINE IS ACCESSED. IF IBF=0, THEN OBF	
		548 ;IS TESTED. IF OBF=1, IBF IS TESTED AGAIN. AS SOON AS OBF=0, RXSTS	
		549 ;IS EXAMINED TO SEE IF DATA IS WAITING FOR OUTPUT. WHEN RX DATA	
		550 ;READY IS SET, F0 IS SET AND F1 IS CLEARED, AND THE DATA IS TRANSFERRED	
		551 ;FROM THE RX HOLDING REGISTER INTO DBBOUT AFTER TESTING FOR ERROR	
		552 ;FLAGS. ANY ERROR FLAGS SET ARE TRANSFERRED TO THE STATUS REGISTER.	
		553 ;IF THE I/O FLAG IS SET, THE PORT IS READ AND THE DATA TRANSFERRED TO	
		554 ;DBBOUT.	
		555 ;	
		556 ;*****	
		557 ;	
020C D614		558 MNLOOP: JNIBF MNLP1 ;IF IBF=0, TEST OBF	
020E 7612		559 JF1 CMDJ1 ;IBF=1, TEST F1 FOR COMMAND	
0210 2453		560 JMP DATA ;F1=0, JUMP TO DATA ROUTINE	
0212 2400		561 CMDJ1: JMP CMD ;OUT-OF-PAGE COMMAND JUMP	
0214 860C		562 MNLP1: JOBF MNLOOP ;WAIT UNTIL DBBOUT IS FREE	
0216 C5		563 SEL RB0 ;RXSTS IN RB0	
0217 FB		564 MOV A,RXSTS ;GET RXSTS	
0218 721E		565 JB3 RXRDY ;TEST RX DATA READY FLAG	
021A F23C		566 JB7 IOFLAG ;TEST IO FLAG	
021C 440C		567 JMP MNLOOP ;LOOP	
		568 ;	
		569 ;RX DATA READY - TRANSFER TO DBBOUT	
		570 ;	
021E 85		571 RXRDY: CLR F0 ;SET F0	
021F 95		572 CPL F0	
0220 A5		573 CLR F1 ;RESET F1	
0221 922E		574 JB4 RXF ;CHECK FRAMING ERROR FLAG	
0223 FB		575 RXRDY1: MOV A,RXSTS ;GET RXSTS	
0224 B235		576 JB5 RXD ;CHECK FOR OVERRUN ERROR	
0226 FB		577 RXRDY2: MOV A,RXSTS ;GET RXSTS AGAIN	

APPENDIX C1 (Continued)

LOC	OBJ	SEQ	SOURCE STATEMENT
0227	53C7	578	ANL A, #NOT (DATRDY OR FRAMER OR OVRUN) ; RESET SOME FLAGS
0229	AB	579	MOV RXSTS, A ; RESTORE RXSTS
022A	FC	580	MOV A, R4 ; GET DATA FROM HOLDING REG
022B	02	581	OUT DBB, A ; PUT IN DBBOUT
022C	440C	582	JMP MNLOOP ; LOOP
		583 ;	
		584	; FRAMING ERROR FLAG SET
		585 ;	
022E	FF	586	RXF: MOV A, R7 ; GET STS
022F	4340	587	ORL A, #FESTS ; SET FRAMING ERROR FLAG
0231	AF	588	MOV R7, A ; RESTORE STS
0232	90	589	MOV STS, A ; LOAD STATUS
0233	4423	590	JMP RXRDY1 ; CONTINUE
		591 ;	
		592	; OVERRUN ERROR FLAG SET
		593 ;	
0235	FF	594	RX0: MOV A, R7 ; GET STS
0236	4380	595	ORL A, #OVSTS ; SET OVERRUN ERROR FLAG
0238	AF	596	MOV R7, A ; RESTORE STS
0239	90	597	MOV STS, A ; LOAD STATUS
023A	4426	598	JMP RXRDY2 ; CONTINUE
		599 ;	
		600	; IO FLAG SET - TEST DIRECTION
		601 ;	
023C	FB	602	IOFLAG: MOV A, RXSTS ; GET RXSTS
023D	D20C	603	JB6 MNLOOP ; PORT IS OUTPUT - NO ACTION
023F	85	604	CLR F0 ; RESET F0
0240	A5	605	CLR F1 ; SET F1
0241	B5	606	CPL F1
0242	537F	607	ANL A, #NOT IOFLG ; RESET IO FLAG
0244	AB	608	MOV RXSTS, A ; RESTORE RXSTS
0245	09	609	IN A, P1 ; READ PORT 1
0246	02	610	OUT DBB, A ; PUT DATA IN DBBOUT
0247	440C	611	JMP MNLOOP ; LOOP
		612 ;	
		613	END

USER SYMBOLS

ASCHSK	007F	B110	013E	B1200	014A	B300	0142	B600	0146	BFFLG	0004	CMD	0100	CMD1	0112
CMD2	0121	CMDJ1	0212	DATA	0153	DATRDY	0008	ERROR	013A	ERRST	0131	FESTS	0040	FRAMER	0010
INIT	0200	INT1	000B	IODATA	0167	IODIR	0040	IOER	0127	IOFLAG	023C	IOFLG	0000	MARK	0004
MKOUT	0001	MNLOOP	020C	MNLP1	0214	MOUT	0019	OVRUN	0020	OVSYS	0000	RCV	001B	RCV1	0026
RCV2	0033	RCV3	0038	RCV4	0042	RCV5	004D	RCV6	0051	RCV7	0055	RCV8	0060	RCV9	0064
REQFLG	0002	RESET	0000	RETURN	0000	RSTERR	003F	RXF	022E	RXFLG	0001	RXINTL	0000	RX0	0235
RXRDY	021E	RXRDY1	0223	RXRDY2	0226	RXSTS	0003	SBIT	0008	SPACE	00FB	SPOU1	00FE	SRTBIT	0003
SRTFLG	0002	STPEND	0024	STTIMR	014C	TICOUT	0040	TICSRT	0004	TINCON	0020	TIMINT	0007	1XBIT	0020
TXEND	0028	TXFLG	0001	TXINT	0008	TXOUT	0012	TXSTS	0005	1XTIC	0003	XMIT	0068	XMT1	0001
XMT2	000C	XMT3	0097	XMT4	009C	XMT5	00A8	ZERO	0000						

ASSEMBLY COMPLETE, NO ERRORS

LOC	DATA	SOURCE	STATUS
4000	4000	4000	4000
4001	4001	4001	4001
4002	4002	4002	4002
4003	4003	4003	4003
4004	4004	4004	4004
4005	4005	4005	4005
4006	4006	4006	4006
4007	4007	4007	4007
4008	4008	4008	4008
4009	4009	4009	4009
4010	4010	4010	4010
4011	4011	4011	4011
4012	4012	4012	4012
4013	4013	4013	4013
4014	4014	4014	4014
4015	4015	4015	4015
4016	4016	4016	4016
4017	4017	4017	4017
4018	4018	4018	4018
4019	4019	4019	4019
4020	4020	4020	4020
4021	4021	4021	4021
4022	4022	4022	4022
4023	4023	4023	4023
4024	4024	4024	4024
4025	4025	4025	4025
4026	4026	4026	4026
4027	4027	4027	4027
4028	4028	4028	4028
4029	4029	4029	4029
4030	4030	4030	4030
4031	4031	4031	4031
4032	4032	4032	4032
4033	4033	4033	4033
4034	4034	4034	4034
4035	4035	4035	4035
4036	4036	4036	4036
4037	4037	4037	4037
4038	4038	4038	4038
4039	4039	4039	4039
4040	4040	4040	4040
4041	4041	4041	4041
4042	4042	4042	4042
4043	4043	4043	4043
4044	4044	4044	4044
4045	4045	4045	4045
4046	4046	4046	4046
4047	4047	4047	4047
4048	4048	4048	4048
4049	4049	4049	4049
4050	4050	4050	4050
4051	4051	4051	4051
4052	4052	4052	4052
4053	4053	4053	4053
4054	4054	4054	4054
4055	4055	4055	4055
4056	4056	4056	4056
4057	4057	4057	4057
4058	4058	4058	4058
4059	4059	4059	4059
4060	4060	4060	4060
4061	4061	4061	4061
4062	4062	4062	4062
4063	4063	4063	4063
4064	4064	4064	4064
4065	4065	4065	4065
4066	4066	4066	4066
4067	4067	4067	4067
4068	4068	4068	4068
4069	4069	4069	4069
4070	4070	4070	4070
4071	4071	4071	4071
4072	4072	4072	4072
4073	4073	4073	4073
4074	4074	4074	4074
4075	4075	4075	4075
4076	4076	4076	4076
4077	4077	4077	4077
4078	4078	4078	4078
4079	4079	4079	4079
4080	4080	4080	4080
4081	4081	4081	4081
4082	4082	4082	4082
4083	4083	4083	4083
4084	4084	4084	4084
4085	4085	4085	4085
4086	4086	4086	4086
4087	4087	4087	4087
4088	4088	4088	4088
4089	4089	4089	4089
4090	4090	4090	4090
4091	4091	4091	4091
4092	4092	4092	4092
4093	4093	4093	4093
4094	4094	4094	4094
4095	4095	4095	4095
4096	4096	4096	4096
4097	4097	4097	4097
4098	4098	4098	4098
4099	4099	4099	4099
4100	4100	4100	4100

Appendix C2

APPENDIX C2

LOC	OBJ	SEQ	SOURCE STATEMENT
		1 ;	
		2 ;	TEST ROUTINE WHICH OUTPUTS THE ASCII CHARACTER SET TO THE
		3 ;	UPI TRANSMITTER AND DISPLAYS ON THE 80/30 CONSOLE ANY
		4 ;	CHARACTERS RECEIVED BY THE UPI RECEIVER.
		5 ;	
		6 ;	INPUTS: NOTHING
		7 ;	OUTPUTS: CHARACTERS TO CONSOLE
		8 ;	CALLS: NOTHING
		9 ;	
4000		10	ORG 4000H
00DF		11	MODE53 EQU 0DFH ; 8253 CONTROL PORT
00DC		12	CNT0 EQU 0DCH ; 8253 CNT 0 PORT
00E5		13	CMD EQU 0E5H ; UPI COMMAND PORT
00E5		14	STATUS EQU 0E5H ; UPI STATUS PORT
00E4		15	DBBIN EQU 0E4H ; UPI DBBIN PORT
00E4		16	DBBOUT EQU 0E4H ; UPI DBBOUT PORT
0020		17	TXINT EQU 20H ; TXINT MASK
0001		18	OBF EQU 01H ; OBF MASK
0002		19	IBF EQU 02H ; IBF MASK
00ED		20	STAT51 EQU 0EDH ; 8251 STATUS PORT
00EC		21	DAT51 EQU 0ECH ; 8251 DATA PORT
0001		22	TXRDY EQU 01H ; 8251 TXRDY MASK
		23 ;	
4000 3E36		24	START: MVI A, 36H ; 8253 CNT0 MODE WORD
4002 D3DF		25	OUT MODE53 ; 8253 CONTROL PORT
4004 3E10		26	MVI A, 10H ; DIVIDE BY 160
4006 D3DC		27	OUT CNT0 ; 8253 CNT0 PORT LSB
4008 3E00		28	MVI A, 00H ;
400A D3DC		29	OUT CNT0 ; 8253 CNT0 PORT MSB
400C 0620		30	MVI B, 20H ; INITIALIZE OUTPUT CHR
400E 3E10		31	MVI A, 10H ; CONFIGURE COMMAND - 1200 BAUD
4010 D3E5		32	OUT CMD ; UPI COMMAND PORT
4012 DBE5		33	POLL1: IN STATUS ; READ UPI STATUS
4014 E621		34	ANI TXINT OR OBF ; TEST TXINT AND OBF
4016 CA1240		35	JZ POLL1 ; WAIT UNTIL ONE IS SET
4019 DBE5		36	IN STATUS ; READ UPI STATUS AGAIN
401B E601		37	ANI OBF ; WAS IT OBF?
401D C23840		38	JNZ RX ; YES, GO DO RECEIVER
		39	; NO, MUST BE TRANSMITTER
4020 78		40	MOV A, B ; GET NEXT CHR FOR OUTPUT
4021 D3E4		41	OUT DBBIN ; OUTPUT TO UPI DBBIN
4023 FE5A		42	CPI 'Z' ; WAS IT LAST CHR?
4025 CA3340		43	JZ NEWB ; YES, RESET REG. B
4028 04		44	INR B ; OTHERWISE, INC B
4029 DBE5		45	POLL2: IN STATUS ; TEST IF IBF STILL SET
402B E602		46	ANI IBF ; TEST IBF
402D C22940		47	JNZ POLL2 ; WAIT UNTIL IBF=0
4030 C31240		48	JMP POLL1 ; BEFORE LOOKING AT STATUS AGAIN
		49 ;	
4033 0620		50	NEWB: MVI B, 20H ; RESET REG. B
4035 C32940		51	JMP POLL2 ; GO BACK
		52 ;	

APPENDIX C2 (Continued)

LOC	OBJ	SEQ	SOURCE STATEMENT
4038	DBE4	53 RX:	IN DBBOUT ; READ DBBOUT FOR RECEIVED CHR
403A	4F	54	MOV C, A ; SAVE IT IN C
403B	DBED	55 RX1:	IN STAT51 ; READ 8251 STATUS
403D	E601	56	ANI TXRDY ; TEST TXRDY
403F	CA3B40	57	JZ RX1 ; WAIT UNTIL READY
4042	79	58	MOV A, C ; GET CHR
4043	D3EC	59	OUT DATA51 ; OUTPUT CHR TO CONSOLE
4045	C31240	60	JMP POLL1 ; GO TEST UPI AGAIN
		61 ;	
		62	END

PUBLIC SYMBOLS

EXTERNAL SYMBOLS

USER SYMBOLS

CMD	A 00E5	CNT0	A 00DC	DATA51	A 00EC	DBBIN	A 00E4	DBBOUT	A 00E4	16F	A 0002	MODE53	A 00DF
NEWB	A 4033	0BF	A 0001	POLL1	A 4012	POLL2	A 4029	RX	A 4038	RX1	A 4038	START	A 4000
STAT51	A 00ED	STATUS	A 00E5	TXINT	A 0020	TXRDY	A 0001						

ASSEMBLY COMPLETE, NO ERRORS

10-3	Introduction
10-8	The LRC Printer
10-4	Interface Signals
10-8	Timing
10-8	Software
10-9	Details of the Buffer Manager
10-11	Printer Service Routines
10-12	Appendix

Printer Control with the UPI-41™

August 1977

Printer Control with the UPI-41™

Lionel Smith
Microcomputer Applications

UPI-41™

Introduction	10-3
The LRC Printer	10-3
Interface Signals	10-4
Timing	10-8
Software	10-8
Details of the Buffer Manager	10-9
Printer Service Routines	10-11
Conclusion	10-15
Appendix	10-15

INTRODUCTION

The UPI-41 is a low-cost, single-chip microcomputer designed to be used as a universal peripheral interface device in a microcomputer system. The device is based on a completely self-contained 8-bit microcomputer with program memory, data memory, CPU, I/O, event timer, and clock oscillator, in a single 40-pin package. A bus interface is included which enables the UPI-41 to be used as a peripheral controller in MCS-48, MCS-80, MCS-85 and other 8-bit microcomputer families. The device is designed for keyboard scanning, printer control, display multiplexing and similar applications which involve interfacing peripheral devices to microcomputer systems.

The UPI-41 is fabricated with N-channel MOS technology and requires only a single 5-volt supply for operation. It has 1K words of program memory and 64 words of data memory on-chip. Both ROM (8041) and EPROM (8741) versions are available and the two are completely pin compatible. The instruction set of the UPI-41 is almost identical to that of the MCS-48. A single byte data register on the UPI-41 interfaces directly to an 8-bit master processor bus to handle asynchronous data transfer to and from the master system. A separate 4-bit register is used to indicate the status of data transfer. Two 8-bit TTL-compatible I/O ports plus two single-bit test inputs are available. I/O can be expanded further by using the 8243 I/O expander device. A separate register in the UPI-41 is used as an event counter or interval timer.

Because it is a complete microcomputer, the UPI-41 provides more power and flexibility than conventional LSI interface devices. For instance, the UPI-41 can be programmed as a peripheral interface for any of the low-cost drum or dot matrix printers currently on the market. In addition to controlling the printer, the UPI-41 can handle zero suppression, limit-checking, formatting and other computations, thereby unburdening the master processor. This type of distributed intelligence, made possible by the UPI-41, greatly enhances overall system capability while reducing cost and development time.

This application note describes how the UPI-41 can be used to implement an interface to a matrix printer. The printer chosen is fairly typical of a large class of printers which minimize total system cost by reducing the mechanical content at the expense of more sophisticated electronic requirements. The UPI-41, with its high degree of capabil-

ity, is ideal for this type of application. It is suggested that the reader not already familiar with the UPI-41 read the "Intel UPI-41 User's Manual" before proceeding in this document.

THE LRC PRINTER

The LRC Model 7040 printer is a matrix printer manufactured by LRC Inc. of Riverton, Wyoming. Capable of printing up to 40 columns of alphanumeric information, this printer is mechanically simple and should be ideal for a variety of applications such as point of sale terminals and data logging. While this note concentrates on the Model 7040 printer, the techniques discussed should be applicable to a variety of similar printers which are currently available.

The printer (Figure 1) consists of four major sub-assemblies, the frame, the print head, the main drive, and the paper handling components. The frame is an aluminum extrusion which provides a suitable base for mounting the various components of the printer. The print head consists of seven solenoids which each drive stiff wires to impact the paper through the inked ribbon. At the solenoid end of the print head these wires are arranged in a circular fashion. Where these wires impact the printer, however, the wires are arranged in a vertical column. To see how this arrangement can be used to print alphanumeric characters refer to Figure 2. The figure shows a 5×7 matrix of "dots". The columns are labeled C1 through C5; the rows are labeled as Row 1 through Row 7. Each row corresponds to one of the solenoid-driven wires. The entire print head assembly is moved left to right across the paper so that at T_1 it is over C1, at T_2 it is over C2, and so on. If the correct solenoids are activated at each of these times (T_1 – T_5) then a character can be formed. Figure 2 shows the character "A" formed. At T_1 solenoids one through five were active, at T_2 solenoids four and six were active, and so on until the complete character was formed. The complete character is formed by choosing the correct pattern of active solenoids for each of five instants in time.

The print head is moved across the paper by the main drive. The main drive consists of a 24-pole synchronous motor which drives a rotating plastic drum. The drum has a spiral groove molded into it. A pin attached to the print head rests in this groove so that as the drum rotates at a constant speed the print head is driven back and forth across the paper. Printing is accomplished by controlling

the activation of the solenoids as the print head is driven from left to right across the paper. When the end of the print area occurs the spiral groove reverses the direction of the head motion. As the left-hand edge of the paper is reached a cam attached to the drum activates the HOME micro-switch and the groove again reverses the motion of the head. When the print head is again over the print area and travelling in the left to right direction the microswitch is deactivated. The printer controller uses the trailing edge of the signal generated by the microswitch to initiate the printing of a new line of information.

Paper feed is accomplished by a second synchronous motor which can be activated to feed paper through the mechanism. A switch is provided which is activated while the actual line feed is occurring. The control logic can use the trailing

edge of the signal generated by this switch to turn off the line feed motor. A version of the printer with automatic line feed is available.

INTERFACE SIGNALS

The interface signals to the printer consists of a pair of wires for each solenoid, a pair of wires for each motor (main drive and line feed), a pair of wires returning the state of the HOME micro-switch, and a pair of wires returning the state of the LINEFEED microswitch.

The solenoids must be driven from a 40 ± 4 volt source. The peak current is approximately 3.6A, the average current is approximately 0.5A. A circuit providing the required drive is shown in Figure 3. The output stage, consisting of the 2N6045 Darlington transistor, the 1N4002 catching diode, and the 20-ohm damping resistor, is the

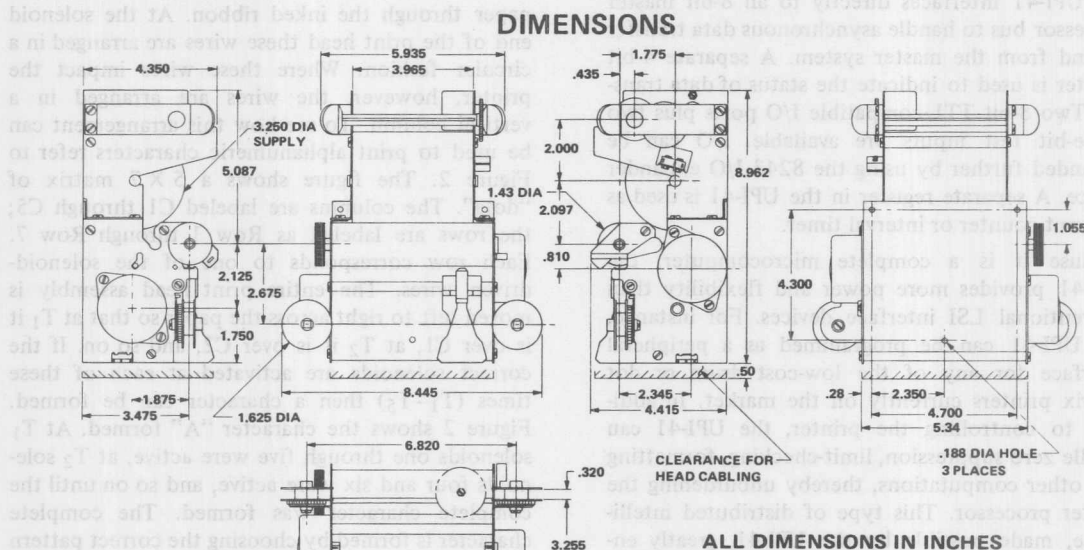


Figure 1. LRC Model 7040 Printer

one suggested by the manufacturer of the printer. The input stage is a discrete implementation of a DTL gate. Note that the base-emitter junction of the 2N6045 will protect the 2N2222A transistor from over-voltage on its collector. This circuit has several features which are important to the printer interface:

1. All solenoid power (including the power used to drive the base of the power transistor) is derived from the 40-volt supply.
2. Disconnecting the drivers from the UPI-41 or the loss of the 5-volt supply to the UPI-41 will result in the solenoids being turned off.

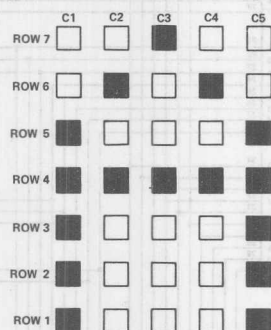


Figure 2. 5 x 7 Dot Matrix

The first feature of the drivers will minimize the impact of the printer and its interface on the 5-volt supply of the system. The second feature prevents the activation of the solenoids erroneously during power on/off cycles or during system checkout. This is an important point since the solenoids will be damaged if left activated continuously. (During the debug of the design described in this note fuses were added to the solenoid drivers to protect them from mishap.)

The two motors can each be driven as shown in Figure 4. The Monsanto MCS-6200 is an optically-coupled TRIAC which is ideal for driving the small synchronous motors in the printer. Coupled with a buffer this part provides a simple means of controlling the motor without sacrificing the isolation required for safe and reliable operation.

Figure 5 shows a UPI-41 used as an interface between an Intel® 8085 and an LRC Model 7040 printer. The drivers which have already been described have been used to interface the TTL outputs of the 8741 to the levels required by the printer. The two contact closure outputs from the printer (PAPERFEED and HOME) have been filtered and applied to the TEST0 and TEST1 inputs of the UPI-41. Bit 5 of output port 2 has been designated as an interrupt pin which will be used to request service from the 8085.

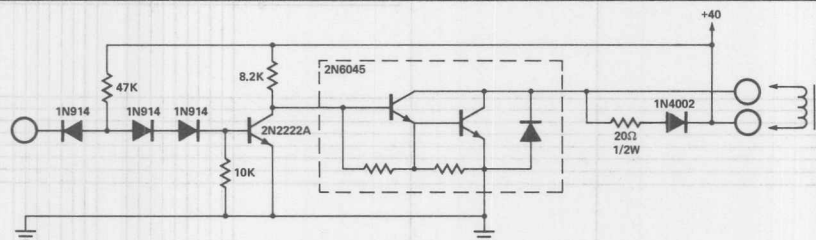


Figure 3. Solenoid Driver

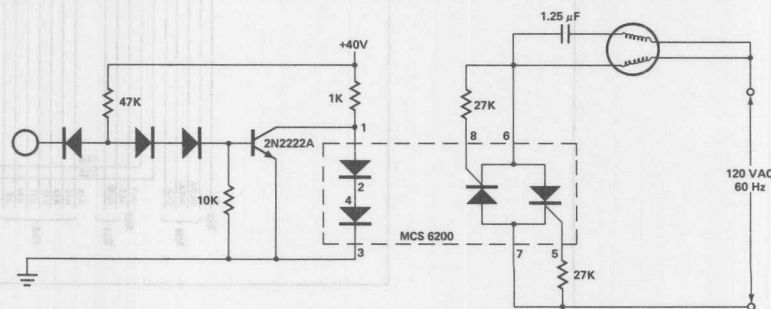


Figure 4. Motor Driver

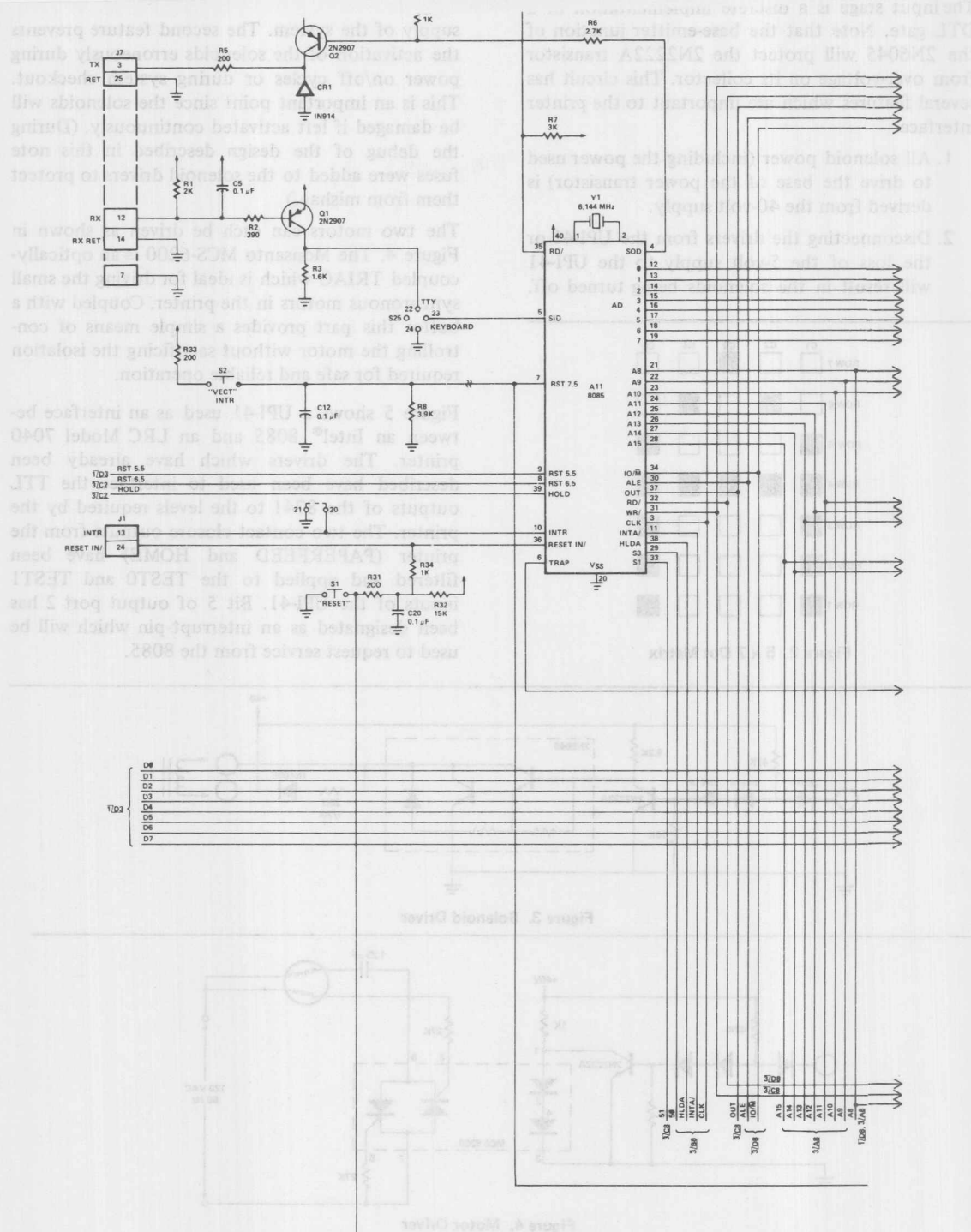
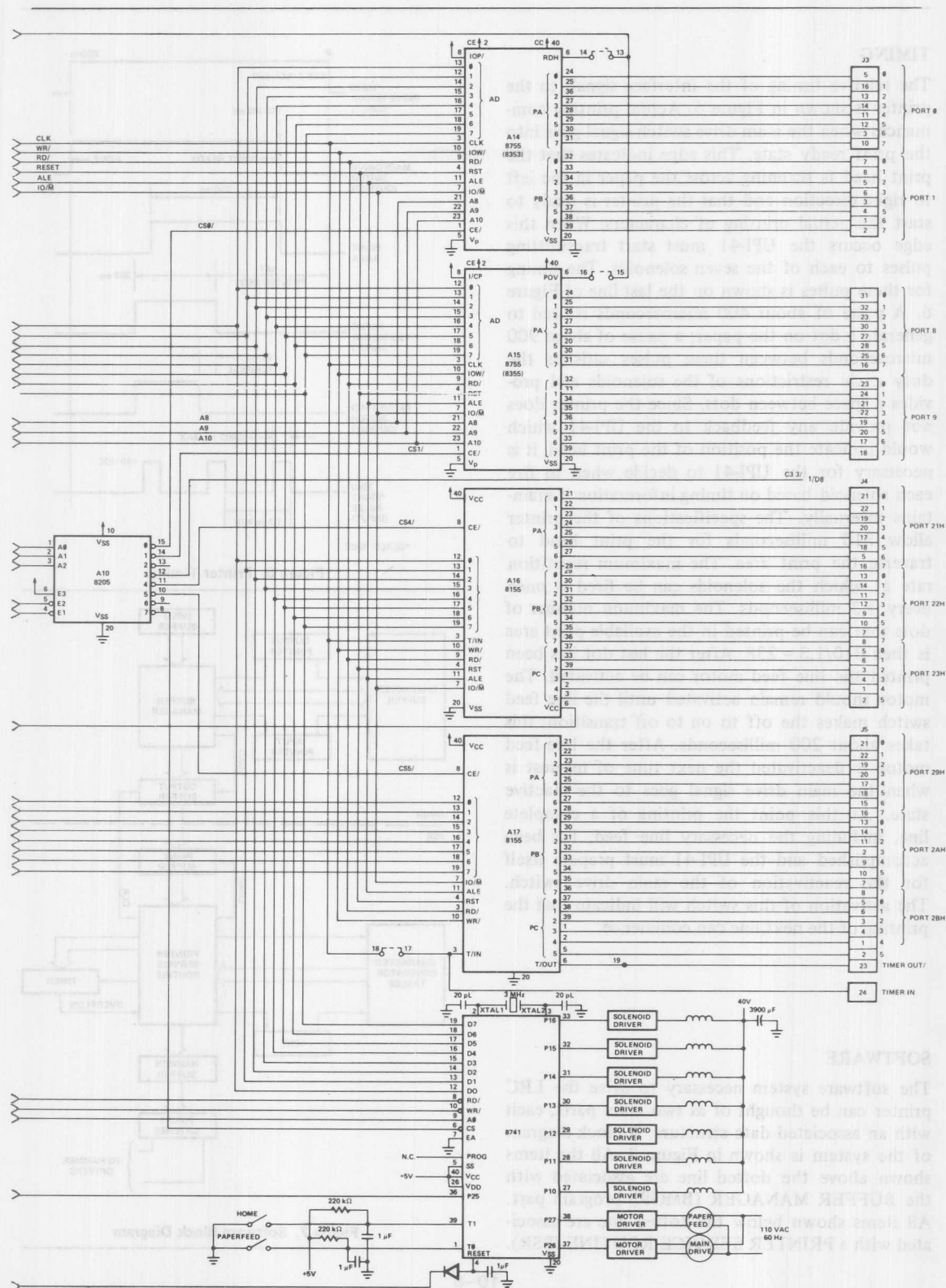


Figure 5. SDK-85 + UPI-41



TIMING

The relative timing of the interface signals to the printer is shown in Figure 6. Actual printing commences when the main drive switch signal goes into the print ready state. This edge indicates that the print head is scanning across the paper in the left to right direction and that the printer is ready to start the actual printing of characters. When this edge occurs the UPI-41 must start transmitting pulses to each of the seven solenoids. The timing for these pulses is shown on the last line of Figure 6. A pulse of about 400 microseconds is used to generate a dot on the paper; a pause of about 900 microseconds between these pulses satisfies the duty cycle restrictions of the solenoids and provides a space between dots. Since the printer does not provide any feedback to the UPI-41 which would indicate the position of the print head, it is necessary for the UPI-41 to decide when to fire each solenoid based on timing information it maintains internally. The specifications of the printer allow 310 milliseconds for the print head to traverse the print area. The maximum repetition rate at which the solenoids can be fired is once every 1.3 milliseconds. The maximum number of dots that can be printed in the available print area is then $310/1.3 = 238$. After the last dot has been printed the line feed motor can be activated. The motor should remain activated until the line feed switch makes the off to on to off transition; this takes about 200 milliseconds. After the line feed motor is deactivated the next time of interest is when the main drive signal goes to the inactive state. At this point the printing of a complete line, including the necessary line feed, has been accomplished and the UPI-41 must prepare itself for the reactivation of the main drive switch. The activation of this switch will indicate that the printing of the next line can commence.

SOFTWARE

The software system necessary to drive the LRC printer can be thought of as two main parts, each with an associated data structure. A block diagram of the system is shown in Figure 7. All the items shown above the dotted line are associated with the BUFFER MANAGER (BMGR) program part. All items shown below the dotted line are associated with a PRINTER SERVICE ROUTINE (PSR).

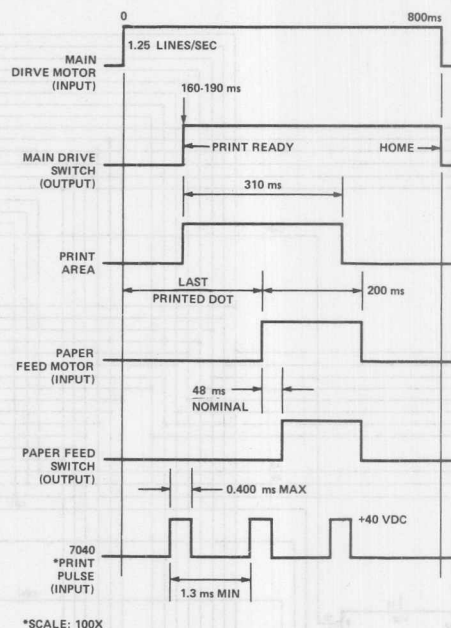


Figure 6. Printer Timing

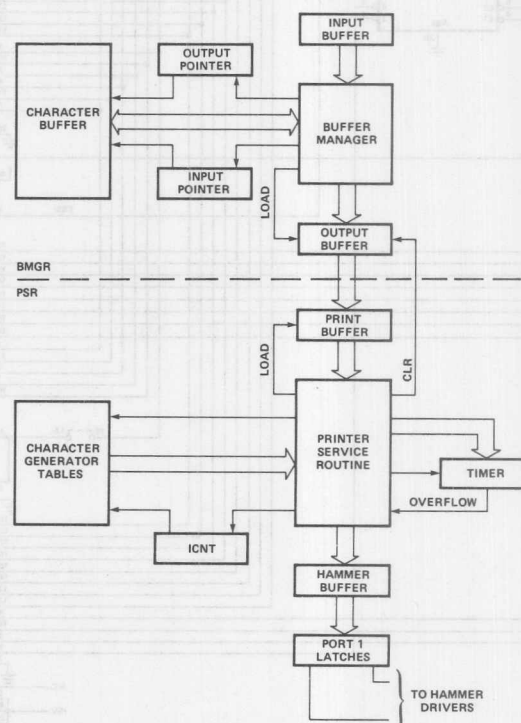


Figure 7. Software Block Diagram

The BUFFER MANAGER is responsible for all interaction with the master processor (i.e., the 8085 in Figure 5). The data structure associated with BMGR is a 40-character buffer which is used to store the characters as they are received from the master processor. BMGR maintains two pointers which are used to access the buffer; these pointers are shown as INPUT POINTER and OUTPUT POINTER in the diagram and are implemented as UPI-41 registers R₀ and R₁, respectively. The input pointer (INPNT) is kept pointing to the last character loaded into the buffer, the output pointer (OUTPNT) is kept pointing to the next character to be printed. BMGR has two major interfaces, the INPUT BUFFER, which is used to communicate with the master processor, and the register shown in the figure as OUTPUT BUFFER. This register, which is implemented with register R₃ of the UPI-41, is used to communicate with the printer service routine (PSR). A character to be printed is placed in the output buffer (OBUF). When PSR is ready to print the character it moves it from OBUF to its own buffer (PBUF) which is labeled as PRINT BUFFER in the diagram. After the character is moved the output buffer is overwritten by a predetermined value which indicates that PSR has accepted the character. BMGR will load a character into the output buffer only if it currently is equal to this value.

The printer service routine utilizes the TIMER to keep track of the current position of the print head. At the appropriate times it causes the solenoid drivers to be pulsed so that the character stream it sees in PBUF is printed. Based on the contents of PBUF and the contents of ICNT, which indicates the active column of the current character, PSR looks up the appropriate column data to be printed in the character generator tables. This data is stored in the HAMMER BUFFER until the precise time that it should be presented to the hammer drivers via the I/O bits in PORT 1. ICNT and the HAMMER BUFFER are implemented as UPI-41 registers 5 and 7, respectively.

DETAILS OF THE BUFFER MANAGER

Before BMGR can be discussed in detail, the manner in which it utilizes the character buffer must be understood. Figure 8 shows the operation of the buffer while two lines of data are input to the UPI-41 and subsequently printed. In order to keep the discussion manageable, this figure is drawn as if the printer were capable of printing only four

characters per line. The two lines of characters to be printed are:

ABCD
1234

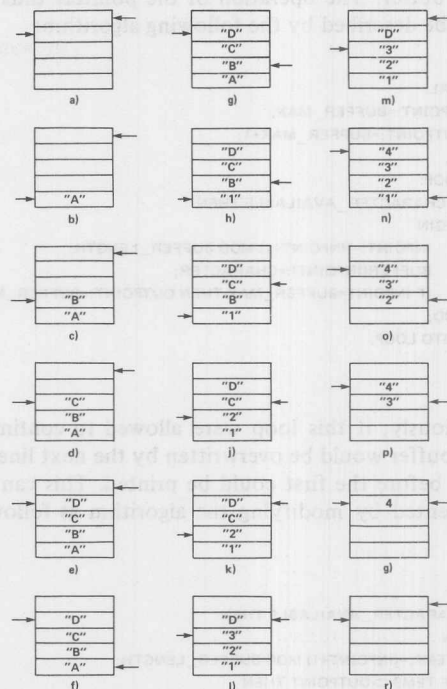


Figure 8. Buffer Operation

It should be noted that the buffer contains 5 bytes, one more than the number of print positions. The extra byte is a "phantom address" which, when pointed to by the output pointer, indicates that the section of BMGR which services the printer service routine is inactive. This state must be allowed because the actual print operation cannot begin until the complete line has been input to the buffer. If this rule were not enforced, some under-run protocol would have to be established to handle the situation of the input stream from the master processor failing to keep up with the print head.

Figure 8a shows the buffer in its initial state. The input pointer is set to the last real position in the buffer and the output pointer is set to the phantom position. Figures 8b through 8f show the operation of the pointers as the characters "A", "B", "C", and "D" are loaded. In each case the

input pointer is incremented to point to the next available location and then that location is loaded with the character. The position of the output pointer is not changed until the last position of the buffer has been loaded. When this occurs, the output pointer is set to point at the first character of the buffer. The operation of the pointers thus far can be described by the following algorithm:

```
INITIAL:
  INPOINT:=BUFFER_MAX;
  OUTPOINT:=BUFFER_MAX+1;
  ...
LOOP:
  IF CHARACTER_AVAILABLE THEN
  BEGIN
    INPOINT:=(INPOINT+1) MOD BUFFER_LENGTH;
    BUFFER(INPOINT):=CHARACTER;
    IF INPOINT=BUFFER_MAX THEN OUTPOINT:=BUFFER_MIN;
  END;
  GOTO LOOP;
END;
```

Obviously, if this loop were allowed to continue, the buffer would be overwritten by the next line of text before the first could be printed. This can be prevented by modifying the algorithm as follows:

```
...
LOOP:
  IF CHARACTER_AVAILABLE THEN
  BEGIN
    TEMP:=(INPOINT+1) MOD BUFFER_LENGTH;
    IF TEMP<>OUTPOINT THEN
    BEGIN
      INPOINT:=TEMP;
      BUFFER(INPOINT):=CHARACTER;
      IF INPOINT=BUFFER_MAX THEN OUTPOINT:=BUFFER_MIN;
    END;
  END;
  GOTO LOOP;
```

This modification will "freeze the action" at Figure 8f until the output pointer is incremented. When this occurs the input procedure will immediately load the input data over the character that was just printed (assuming that data is available to the procedure at a higher rate than can be printed). The defined interface with the printer service routine allows a character to be removed from the buffer and placed in the output buffer whenever the output buffer contains the value placed there by the PSR, indicating that it has accepted the character that was previously in the output buffer. If this value is called EMPTY_FLAG then the complete buffer handling procedure can be defined as follows:

```
INITIAL:
  INPOINT:=BUFFER_MAX;
  OUTPOINT:=BUFFER_MAX+1;
  ...
LOOP:
  IF CHARACTER_AVAILABLE THEN
  BEGIN
    TEMP:=(INPOINT+1) MOD BUFFER_LENGTH;
    IF TEMP<>OUTPOINT THEN
    BEGIN
      INPOINT:=TEMP;
      BUFFER(INPOINT):=CHARACTER;
      IF INPOINT=BUFFER_MAX THEN
        OUTPOINT:=BUFFER_MIN;
    END;
    IF OUTPUT_BUFFER=EMPTY_FLAG THEN
    BEGIN
      IF OUTPOINT<=BUFFER_MAX THEN
      BEGIN
        OUTPUT_BUFFER:=BUFFER(OUTPOINT);
        OUTPOINT:=OUTPOINT+1;
      END;
    END;
  END;
  GOTO LOOP;
```

Examination of Figures 8g through 8r will show how this algorithm maintains the buffer. If there is an open position and a character is available, it is placed in the buffer. When a complete line is in the buffer, printing is initialized by setting the output pointer to BUFFER_MIN. As the last character of a line is printed, the output pointer is incremented to point at the "phantom location" until the next line is completely entered. It should also be noted that if the input stream is faster than the print operation, then after the last character of a line is printed only one character need be input before printing can resume (see Figures 8l, m, and n). Frame r shows that after all available characters have been printed the state of the buffer is the same as it is initially. This is obviously a desirable feature.

The flowcharts for the complete BUFFER MANAGER are shown in Figures 9a and 9b. The corresponding code can be found starting at label BMGR of the program listings (see appendix). The flowcharts follow the algorithm that has been discussed very closely. Some additions have been made to implement logic not associated with the buffer. The first difference is that when a byte is in the input buffer it is tested to determine whether it is a command byte or a data character before further action is taken. Only two commands are recognized; one to set, and one to reset, the internal interrupt enable flag. This flag, which is

implemented as bit zero of PORT2 determines whether or not the UPI-41 will assert an interrupt to the master processor when it is able to accept a new character. Two additional deviations can be noted in Figure 9a; the first is that the motor of the printer will be turned on whenever a data character is received, the second is that if an end of line code (i.e., an ASCII line feed) is received, then, instead of storing it in the buffer, a mode is entered which fills the remaining buffer locations with space characters. This mode is enabled by bit one of PORT2. Note that utilizing otherwise unused bits of PORT2 for program status allows convenient testing and setting by the software and also enables external monitoring of the program operation.

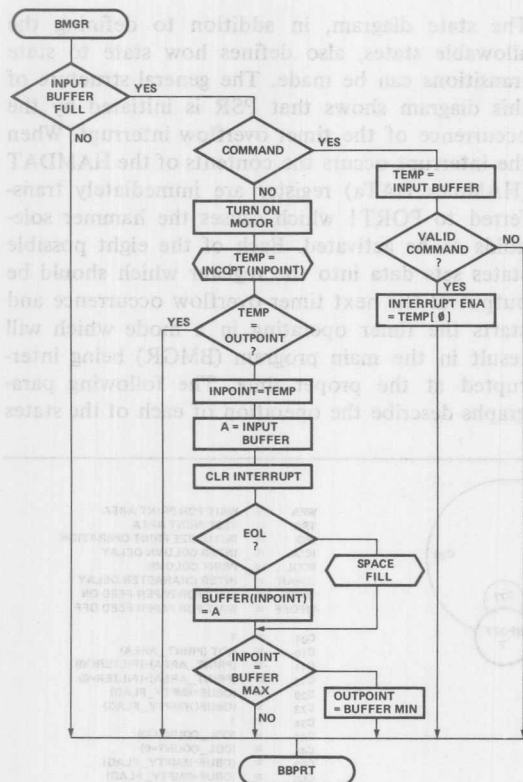


Figure 9a. Buffer Manager Flowchart

The last addition to the algorithm can be seen in Figure 9b where instead of going directly back to the start of the program after servicing the printer, a test is made to determine if the interrupt to the master processor should be asserted. This interrupt is set if the enable bit is set and there is also room in the buffer for at least one more character. After this test, control is passed back to the beginning of BMGR.

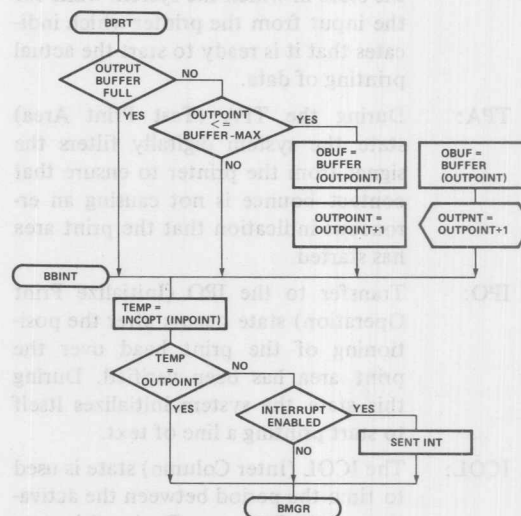


Figure 9b. Buffer Manager Flowchart

PRINTER SERVICE ROUTINES

The Printer Service Routine must convert the characters given to it by the Buffer Manager into an appropriately timed stream of pulses to the solenoids. Because the PSR is extremely time-dependent, it was implemented as an interrupt-driven routine which is given control when the timer overflow occurs. This allows exact timing of the solenoid firings without requiring software delay loops. If the timing had been generated by such loops, synchronization would have been lost when the delay loops were interrupted in order to service the master processor.

If a hardware design of a controller for the printer were being undertaken, a convenient place to start would be to generate a state transition diagram which shows all the states that can be entered and how control can transfer from state to state. This hardware design technique is often useful in software design and was, in fact, used to develop the PSR. The state diagram of the PSR is shown in Figure 10. A total of eight states are necessary to implement the printer control function. Before discussing this diagram further, each of these states must be defined.

WPA: The WPA (Wait for Print Area) state is the state in which the system waits for the input from the printer which indicates that it is ready to start the actual printing of data.

TPA: During the TPA (Test Print Area) state the system digitally filters the signal from the printer to ensure that contact bounce is not causing an erroneous indication that the print area has started.

IPO: Transfer to the IPO (Initialize Print Operation) state occurs after the positioning of the print head over the print area has been verified. During this state the system initializes itself to start printing a line of text.

ICOL: The ICOL (Inter Column) state is used to time the period between the activation of the hammers. During this state the space between the dots of the characters is generated.

PCOL: During the PCOL (Print Column) state the hammers are energized if the particular character being printed requires a dot in the corresponding position.

ICHAR: The ICHAR (Inter Character) state is active between characters on a given line.

WFON: During the WFON (Wait for Feed On) state the system waits for the assertion of the feed pulse from the printer. This signal indicates that the process of feeding paper is occurring.

WFOFF: The system remains in the WFOFF (Wait for Feed Off) until the feed pulse goes inactive. This indicates that the required paper feed operation has been completed.

The state diagram, in addition to defining the allowable states, also defines how state to state transitions can be made. The general structure of this diagram shows that PSR is initiated by the occurrence of the timer overflow interrupt. When the interrupt occurs the contents of the HAMDAT (HAMmer DATa) register are immediately transferred to PORT1 which causes the hammer solenoids to be activated. Each of the eight possible states sets data into the register which should be output at the next timer overflow occurrence and starts the timer operating in a mode which will result in the main program (BMGR) being interrupted at the proper time. The following paragraphs describe the operation of each of the states

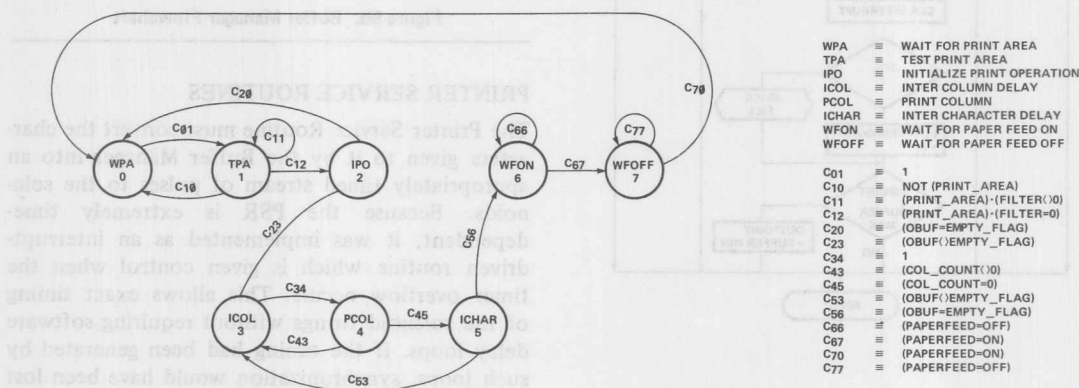


Figure 10. Print Control State Transition Diagram

in detail. The flowcharts of the routines can be found in Figure 11.

The WPA, CPA, and IPO states are all associated with the detection of the valid start of the print area. The WPA state sets the timer in the event count mode so that the edge of the print area signal can be detected, the CPA state digitally filters this input once it has been detected to ensure that noise has not caused a false input, and finally, the IPO state initializes the system to start the actual printing of data. The flowchart shows that the WPA state accomplishes the following actions:

1. Turns off the paper feed motor
2. Sets the filter count (for the CPA state)
3. Sets HAMDAT to zero
4. Sets STATE to one.

The timer is set to event count with an initial value of OFFH. This will cause a timer overflow interrupt the next time a negative transition occurs on the TEST1 input. Since this input is tied to the signal from the PRINT AREA switch, this interrupt should occur when the start of the print area is reached. The WPA state sets the STATE register to cause the TPA state to be entered when this interrupt occurs. Each time the TPA (Test Print Area) state is activated the software checks to ensure that the print area switch is in the proper state; if it is not, then all the actions of state zero are repeated (except turning off the motor), since a false start of print area has occurred. If the test reveals that the print area switch is in the proper state, then the filter count is reduced by one and the timer is started with an initial value of OFFH, the minimum attainable timer increment. The STATE register is set to repeat the TPA state unless the filter count has reached zero; when this occurs the IPO state is selected. The IPO state, which is responsible for the initialization of the actual print operation, first tests the output buffer register to determine if there is any data for it to print. If this test is unsuccessful the printer main drive motor is turned off, the TPA state is reinvoked and the timer is started in the event count mode so that it can detect the next start of print area. At first glance this seems somewhat fruitless since the event required cannot happen if the motor is not turning. By referring back to Figure 9, however, it can be seen that BMGR turns on the motor whenever it has a data character from the master computer. The reception of a character will always allow the PSR to find the next print area. If, when the IPO state makes its

test, there is data in the output buffer then the data is moved to the print buffer and the output buffer is set to the empty value. After this is accomplished, a counter is set to the number of columns to be printed per character (seven in this case — see comment by CGEN label in program listing), the STATE register is set to the ICOL state and the timer is set to time the intercolumn time. (The intercolumn time is the time that elapses between each possible column of the character.) Before exiting from this state the first column of data for the hammers is generated by the COLUMN routine and placed in the HAMDAT register.

The three states already discussed set the printer up so that it is ready to print. The next three states are repeated sequentially until the entire line of data has been printed. The ICOL state is probably the simplest of the states. When it is invoked the hammers have just been fired by the entry into the PSR. All that the ICOL state does is to set the timer to time the proper duration of the hammer strikes, clear the HAMDAT register, and set the STATE register to the PCOL state. The PCOL state, only slightly more complicated than the ICOL state, first decrements the column count. If the end of a character is detected (count equal zero), the HAMDAT register is cleared and the STATE register is set to invoke the ICHAR state. If the end of a character is not detected then the COLUMN routine is again used to determine the next data to be sent to the hammers and the ICOL state is reinvoked. When the ICOL state is active two things can happen, depending on whether there is more data to print. If there is data in the output buffer then a series of actions similar to those of the IPO state occur to reinitialize the printing of a character; if there is no more data in the line then the paper feed motor is turned on, HAMDAT is cleared, and the STATE register is set to the WFON state. The timer is set for approximately one millisecond so that the state of the paper feed switch can be sampled periodically by the WFON and WFOFF states.

The WFON and WFOFF states continue to set the timer to the one millisecond sample rate, the WFON state reinvokes itself until the paper feed switch input is detected and then it invokes the WFOFF state. The WFOFF state reinvokes itself until the paper feed switch is detected in the off state and then invokes the WPA state. The sole purpose of the WFON and WFOFF states is to ensure that an off to on to off transition occurs on

the next line of data.

CONCLUSION

The UPI-41 has been shown to be easily capable of controlling the LRC matrix printer with no external logic other than drivers and receivers. The program listings which implement the algorithms discussed are shown in Appendix A. It should be noted that no attempt has been made to minimize the amount of code in the program; the emphasis

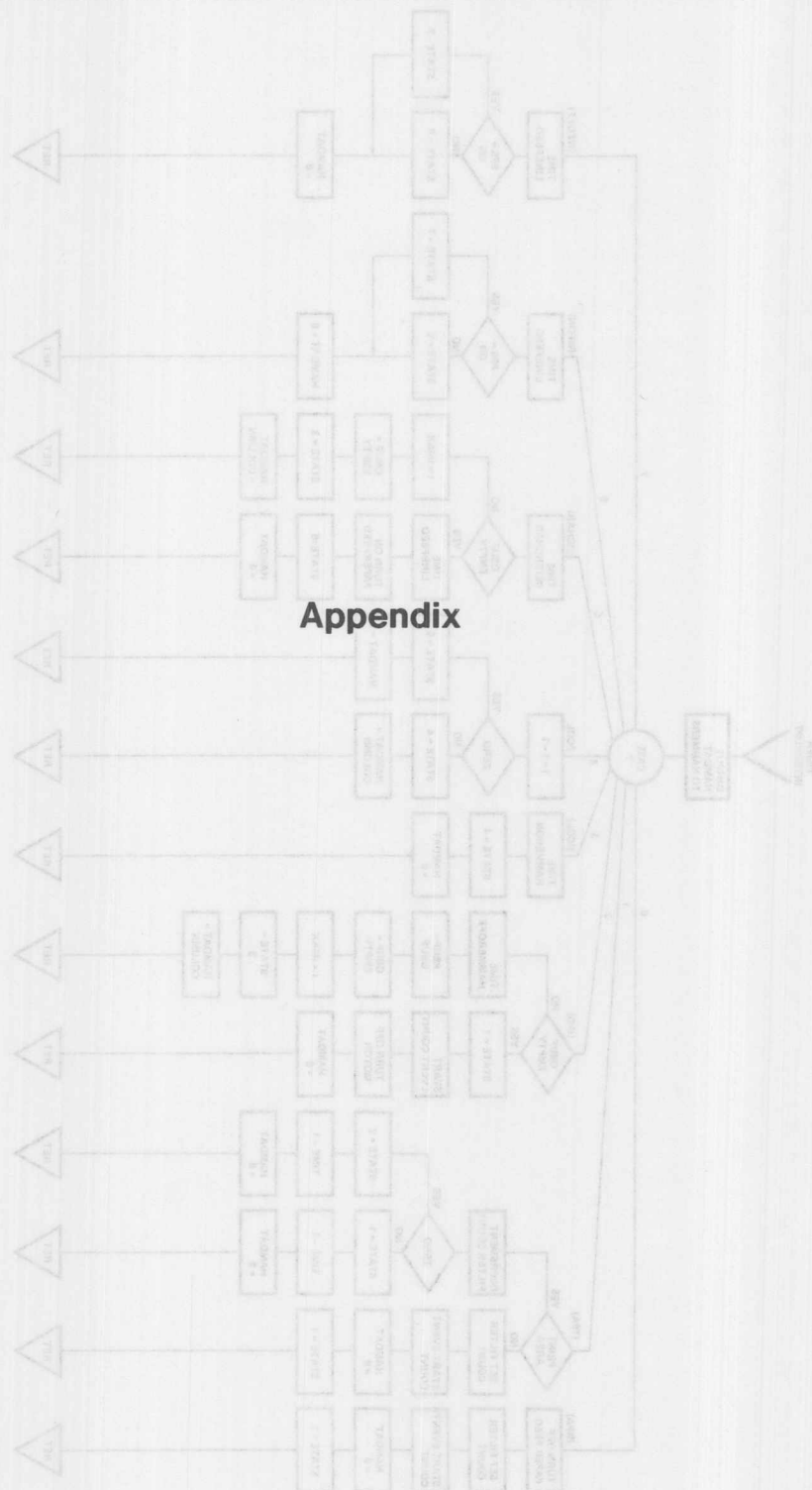
significantly reduce the amount of code space needed, especially in the printer service routine which duplicates much code in each STATE. Even with this relatively loose coding the printer control function, including the complete character tables, easily fit within the memory available in the UPI-41. The extra room in memory could be used to implement such extra features as tabulation, printing prestored messages, or even limited graphic capabilities. The power and flexibility of the UPI-41 make such features easy to implement.

The three states already discussed set the printer up so that it is ready to print. The next three states are repeated repeatedly until the entire line of data has been printed. The ICOL state is probably the simplest of the states. When it is invoked the hammers have just been fired by the entry into the PFR. All that the ICOL state does is to set the timer to time the proper duration of the hammer strikes, clear the HAMDAT register, and set the STATE register to the PCOL state. The PCOL state, only slightly more complicated than the ICOL state, first decrements the column count. If the end of a character is detected (count equal to zero), the HAMDAT register is cleared and the STATE register is set to invoke the NCHAR state. If the end of a character is not detected then the COLUMN routine is again used to determine the next data to be sent to the hammers and the ICOL state is reinvoked. When the ICOL state is active two things can happen, depending on whether there is more data to print. If there is data in the output buffer then a series of actions similar to those of the PFO state occur to reinitiate the printing of a character; if there is no more data in the line then the paper lead motor is turned on, HAMDAT is cleared, and the STATE register is set to the WFO state. The timer is set for approximately one millisecond so that the state of the paper lead switch can be sampled periodically by the WFO and WPOF states.

The WFO and WPOF states continue to set the timer to the one millisecond sample rate, the WFO state reinvokes itself until the paper lead switch input is detected and then it invokes the WPOF state. The WPOF state reinvokes itself until the paper lead switch is detected in the off state and then invokes the WPA state. The sole purpose of the WFO and WPOF states is to ensure that an off to on transition occurs on

1. Turn off the paper lead motor.
2. Set the filter count (for the CPA state).
3. Set HAMDAT to zero.
4. Set STATE to one.

The filter is set to event count with an initial value of 05FH. This will cause a filter overflow interrupt on the next time a negative transition occurs on the TEST1 input. Since this input is tied to the signal from the PRINT AREA switch, this interrupt should occur when the start of the print area is reached. The WPA state sets the STATE register to cause the TPA state to be entered when this interrupt occurs. Each time the TPA (Test Print Area) state is activated the software checks to ensure that the print area switch is in the proper state; if it is not, then all the actions of state zero are repeated (except turning off the motor), since a false start of print area has occurred. If the test reveals that the print area switch is in the proper state, then the filter count is reduced by one and the timer is started with an initial value of 05FH. The minimum attainable timer increment. The STATE register is set to repeat the TPA state unless the filter count has reached zero; when this occurs the PFO state is selected. The PFO state, which is responsible for the initialization of the actual print operation, first tests the output buffer register to determine if there is any data for it to print. If this test is unsuccessful, the printer main drive motor is turned off, the TPA state is reinvoked and the timer is started in the event count mode so that it can detect the next start of print area. At first glance this seems somewhat foolish since the event required cannot happen if the motor is not turning. By referring back to Figure 8, however, it can be seen that BMDR turns on the motor whenever it has a data character from the master computer. The reception of a character will always allow the PFR to find the next print area. If, when the PFO state makes its



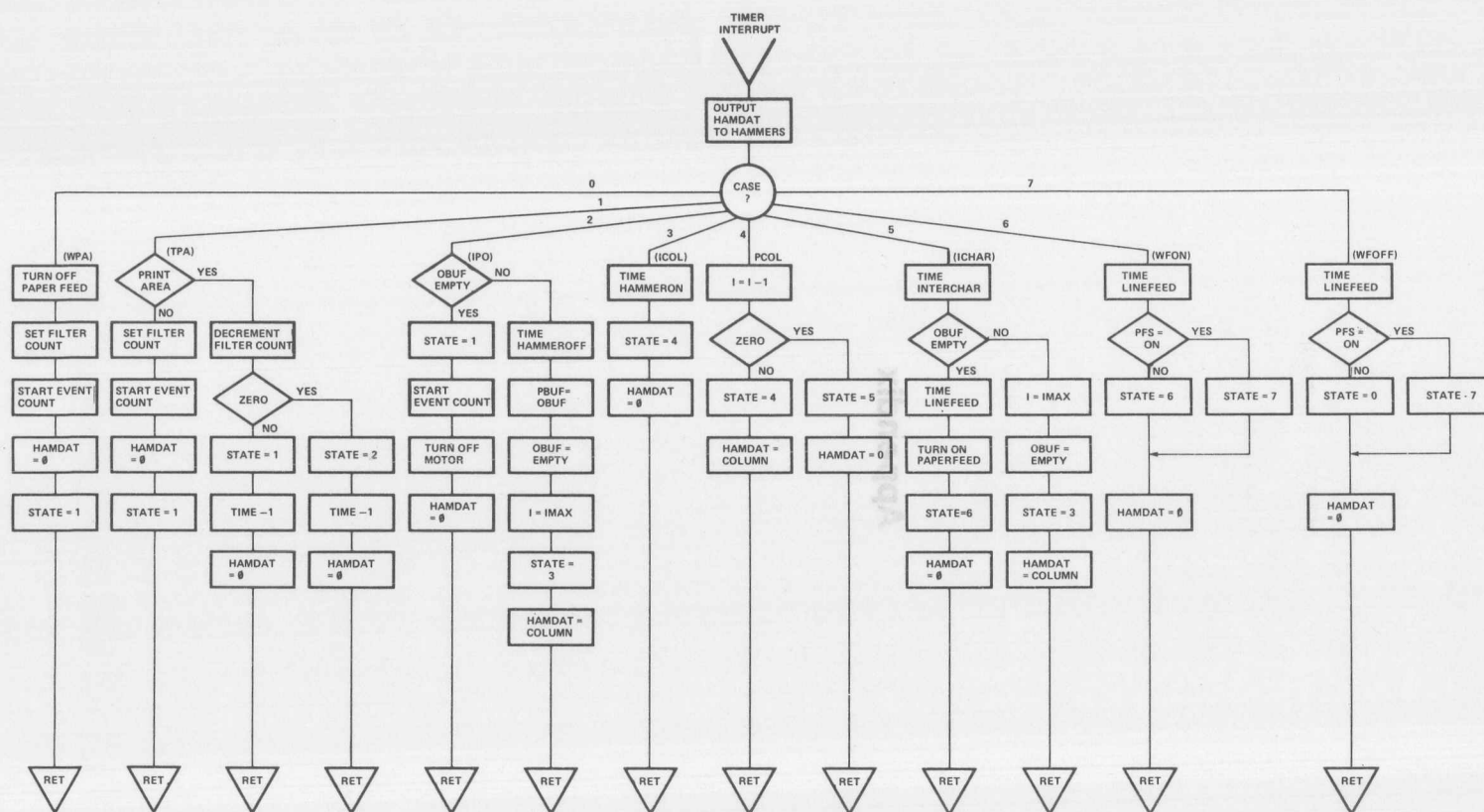


Figure 11. PSR Flowchart

APPENDIX

ISIS-II 8048 ASSEMBLER, V1.1
LRC PRINTER CONTROLLER 7/14/7

LOC	OBJ	SEQ	SOURCE STATEMENT
1			
2			*****
3			;
4			;
5			UPI-41 LRC PRINTER CONTROLLER
6			;
7			THIS PROGRAM IMPLEMENTS THE CONTROL OF THE
8			LRC PRINTER WITH THE UPI-41. DATA IS INPUT TO THE
9			UPI-41 AS SIX BIT ASCII. COMMANDS ARE PROVIDED
10			TO ENABLE OR DISABLE THE GENERATION OF AN
11			INTERRUPT WHEN THE UNIT IS READY
12			FOR ANOTHER DATA CHARACTER. THE INTERRUPT IS ENABLED
13			BY OUTPUTTING 03H TO THE CONTROL CHANNEL AND DISABLED
14			BY OUTPUTTING 02H. WHEN ENABLED THE INTERRUPT
15			IS IMPLEMENTED AS A POSITIVE GOING EDGE ON P25.
16			;
17			NOTE: A PL/M LIKE LANGUAGE WAS USED TO COMMENT
18			THIS PROGRAM. NO COMPILER EXISTS FOR THE UPI-41.
19			THE COMMENTS WERE 'HAND COMPILED' INTO UPI-41
20			ASSEMBLY LANGUAGE.
21			*****
22			;
23			;
24			*****
25			;
26			*****
27			;
28			REGISTER ASSIGNMENTS
29			*****
30			;
31			;
32			;
33	0007		HAMDAT EQU R7
34	0006		STATE EQU R6
35	0005		ICNT EQU R5
36	0004		PBUF EQU R4
37	0003		OBUF EQU R3
38	0002		TESTR EQU R2
39	0001		OUTPNT EQU R1
40	0000		INPNT EQU R0
41			;
42			;
43			*****
44			;
45			TIMER EQUATES
46			*****
47			;
48			*****
49	00A0		TICK EQU 160
50	FFFE		THON EQU -320/TICK
51	FFFD		THOFF EQU -480/TICK
52	FFF8		TINTIR EQU -1280/TICK
53	FFFA		TLFEED EQU -1000/TICK
54	0004		FILTV EQU 640/TICK
55			;
56			*****
57			;
58			*****
59			;
60			PROGRAM MASKS
61			*****
62			;
63			*****
64	00FF		EMTFLG EQU 0FFH
65	0007		IMAX EQU 07H
66	007F		PFED EQU 7FH
67	00BF		MOTON EQU 0BFH
68	0001		INTENA EQU 01H
69	0002		FMODE EQU 02H
70	000A		EOL EQU 0AH
71	0021		EXCLAIM EQU 021H
72	0020		SPACE EQU 20H
73	0020		EXREQ EQU 20H
74	0018		OPTMIN EQU 18H
75	0018		BMIN EQU 18H
76	003F		BMAX EQU 3FH
77			;
78			;
79	\$		EJECT

```

80 *****
81 ;
82 ;
83 START OF PROGRAM
84 ;
85 *****
86 ;
87 ; INITIALIZE;
88 ; INITIALIZE AND GO TO
89 ; BMGR
0000 1416 91 RESET: ORG 00H
0002 3479 92 CALL CASE0
0004 25 93 EN TCNTI
0005 2400 94 JMP BMGR ; CODE MUST END AT LOC 6
95
96 *****
97 ;
98 ;
99 START OF INTERRUPT DRIVEN STATE MACHINE
100 ;
101 *****
102 ; DO;
103 ; HAMMERS=HAMMERS$DAT;
104 ; DO CASE STATE;
105
106 TISR: XCH A,HAMDAT
107 CPL A
108 OUTL P1,A
109 MOV A,STATE
110 ADD A,#CBASE
111 JMPP @A
112 CBASE: DB CASE0
113 DB CASE1
114 DB CASE2
115 DB CASE3
116 DB CASE4
117 DB CASE5
118 DB CASE6
119 DB CASE7
120
121
122
123 ;
124 ; DO; /*CASE 0, FEEDING LINE */
125 ; PAPERSFEED=OFF;
126 ; STATE=1;
127 ; ICNT=FILTIV
128 ; WAIT (PRINT$AREA);
129 ; HAMMERS$DATA=0;
130 ; END; /* END OF CASE 0 */
131
132 CASE0: ORL P2,#(NOT PFEEED)
133 MOV STATE,#1
134 MOV ICNT,#FILTIV
135 MOV A,#-1
136 MOV T,A
137 STRT CNT
138 MOV A,#0
139 XCH A,HAMDAT
140 RETR
141
142 ; DO; /* CASE1, TESTING FOR PRINT AREA */
143 ; IF T0=1 THEN
144 ; DO;
145 ; STATE=1;
146 ; ICNT=FILTIV;
147 ; WAIT (PRINT$AREA);
148 ; HAMMERS$DATA=0;
149 ; END;
150
151 CASE1: JNTI C1ELS
152 MOV STATE,#1
153 MOV ICNT,#FILTIV
154 MOV A,#-1
155 MOV T,A
156 STRT T
157 MOV A,#0
158 JMP C1END

```

LOC	OBJ	SEQ	SOURCE STATEMENT
		158	ELSE DO;
		159	ICNT=ICNT-1;
		160	IF ICNT=0 THEN STATE=2 ELSE STATE=1;
		161	TIME (-1);
		162	HAMMER\$DATA=0;
		163	END;
0032	BE02	164	C1ELS: MOV STATE,#2
0034	ED38	165	DJNZ ICNT,C1LA
0036	BE01	166	MOV STATE,#1
0038	23FF	167	C1LA: MOV A,#-1
003A	62	168	MOV T,A
003B	55	169	STRT T
003C	2300	170	MOV A,#0
		171	
003E	2F	172	C1END: XCH A,HAMDAT
003F	93	173	RETR
		174	
		175	
		176	DO; /*CASE 2, INITIALIZE PRINT OPERATION */
		177	IF OBUF<>EMPTY\$FLAG THEN
		178	DO;
		179	TIME (HAMMER\$OFF);
		180	PBUF=OBUF;
		181	OBUF=EMPTY\$FLAG;
		182	I=IMAX;
		183	STATE=3;
		184	HAMMER\$DATA=COLUMN(PBUF,I);
		185	END;
0040	FB	186	CASE2: MOV A,OBUF
0041	D3FF	187	XRL A,#EMTFLG
0043	C655	188	JZ C2ELS
0045	23FD	189	MOV A,#THOFF
0047	62	190	MOV T,A
0048	55	191	STRT T
0049	FB	192	MOV A,OBUF
004A	AC	193	MOV PBUF,A
004B	BBFF	194	MOV OBUF,#EMTFLG
004D	BD07	195	MOV ICNT,#IMAX
004F	BE03	196	MOV STATE,#3
0051	54E0	197	CALL COLUMN
0053	045F	198	JMP C2END
		199	
		200	
		201	ELSE DO;
		202	STATE=1;
		203	WAIT (PRINT\$AREA);
		204	MOTOR=OFF;
		205	HAMMER\$DATA=0;
		206	END;
0055	BE01	205	C2ELS: MOV STATE,#1
0057	23FF	206	MOV A,#-1
0059	62	207	MOV T,A
005A	45	208	STRT CNT
005B	8A40	209	ORL P2,#NOT MOTON
005D	2300	210	MOV A,#0
		211	
005F	2F	212	C2END: XCH A,HAMDAT
0060	93	213	RETR
		214	
		215	
		216	DO; /*CASE 3, HAMMER OFF CYCLE */
		217	TIME (HAMMER\$ON);
		218	HAMMER\$DATA=0;
		219	STATE=4;
		220	END; /*END OF CASE 3 */
0061	23FE	220	CASE3: MOV A,#THON
0063	62	221	MOV T,A
0064	55	222	STRT T
0065	2300	223	MOV A,#0
0067	BE04	224	MOV STATE,#4
0069	2F	225	XCH A,HAMDAT
006A	93	226	RETR
		227	\$ EJECT

LOC	OBJ	SEQ	SOURCE STATEMENT	
		228		DO; /*CASE 4, PRINTING COL I OF CHAR */
		229		TIME (HAMMER\$OFF);
006B	23FD	230	CASE4: MOV A, #THOFF;	
006D	62	231	MOV T, A;	
006E	55	232	STRT T	
		233		I=I-1;
		234		IF I=0 THEN
		235		DO;
		236		STATE=5;
		237		HAMMER\$DATA=0;
		238		END
006F	ED77	239	DJNZ ICNT, C4ELS	
0071	BE05	240	MOV STATE, #5	
0073	2300	241	MOV A, #0	
0075	047B	242	JMP C4END	
		243		ELSE DO;
		244		STATE=3;
		245		HAMMER\$DATA=COLUMN (PBUF, I);
		246		END;
0077	BE03	247	C4ELS: MOV STATE, #3	
0079	54E0	248	CALL COLUMN	
		249		END; /* END OF CASE 4 */
007B	2F	250	C4END: XCH A, HAMDAT	
007C	93	251	RETR	
		252		DO; /*CASE 5, INTERCHARACTER SPACE */
		253		TIME (INTER\$CHAR);
007D	23F8	254	CASE5: MOV A, #TINTER	
007F	62	255	MOV T, A	
0080	55	256	STRT T	
		257		IF OBUF<>EMPTY\$FLAG THEN
		258		DO;
		259		PBUF=OBUF;
		260		OBUF=EMPTY\$FLAG;
		261		I=IMAX;
		262		STATE=3;
		263		HAMMER\$DATA=COLUMN (PBUF, I);
		264		END;
0081	FB	265	MOV A, OBUF	
0082	D3FF	266	XRL A, #EMTFLG	
0084	C692	267	JZ C5ELS	
0086	FB	268	MOV A, OBUF	
0087	AC	269	MOV PBUF, A	
0088	BBFF	270	MOV OBUF, #EMTFLG	
008A	BD07	271	MOV ICNT, #IMAX	
008C	BE03	272	MOV STATE, #3	
008E	54E0	273	CALL COLUMN	
0090	049C	274	JMP C5END	
		275		ELSE DO;
		276		TIME (LINESFEED);
		277		PAPER\$FEED=ON;
		278		STATE=6;
		279		HAMMER\$DATA=0;
		280		END;
0092	23FA	281	C5ELS: MOV A, #TLFEED	
0094	62	282	MOV T, A	
0095	55	283	STRT T	
0096	9A7F	284	ANL P2, #PFEED	
0098	BE06	285	MOV STATE, #6	
009A	2300	286	MOV A, #0	
		287		END; /* END OF CASE 5 */
009C	2F	288	C5END: XCH A, HAMDAT	
009D	93	289	RETR	
		290		
		291	\$ EJECT	

LOC	OBJ	SEQ	SOURCE STATEMENT
		292	;
		293	DO; /*CASE 6, WAITING FOR FEED ON */
		294	TIME(LINE\$FEED);
		295	IF PFS=1 THEN
		296	DO;
		297	STATE=7;
		298	END;
009E	23FA	298	CASE6: MOV A, #TLFEED
00A0	62	299	MOV T, A
00A1	55	300	STRT T
00A2	26A8	301	JNT0 C6ELS
00A4	BE07	302	MOV STATE, #7
00A6	04AA	303	JMP C6END
		304	;
		305	ELSE DO;
		306	STATE=6;
		307	END;
00A8	BE06	307	C6ELS: MOV STATE, #6
		308	;
		309	HAMMER\$DATA=0;
00AA	2300	309	C6END: MOV A, #0
00AC	2F	310	XCH A, HAMDAT
00AD	93	311	RETR
		312	;
		313	;
		314	;
		315	DO; /*CASE 7, WAITING FOR FEED OFF */
		316	TIME(LINE\$FEED);
		317	IF PFS=0 THEN
		318	DO;
		319	STATE=0;
		320	END;
00AE	23FA	320	CASE7: MOV A, #TLFEED
00B0	62	321	MOV T, A
00B1	55	322	STRT T
00B2	36B8	323	JT0 C7ELS
00B4	BE00	324	MOV STATE, #0
00B6	04BA	325	JMP C7END
		326	;
		327	ELSE DO;
		328	STATE=7;
		329	END;
		330	HAMMER\$DATA=0;
00B8	BE07	331	C7ELS: MOV STATE, #7
00BA	2300	332	MOV A, #0
00BC	2F	333	XCH A, HAMDAT
00BD	93	334	RETR
		335	;
		336	END; /* END OF CASE BLOCK */
		337	;
		338	;
		339	;
		340	;
		341	*****
		342	;
		343	BMGR
		344	;
		345	THIS SEGMENT CONTROLS THE HANDSHAKING BETWEEN THE
		346	CONTROLLER AND THE MASTER PROCESSOR.
		347	;
		348	*****
		349	;
		350	;
		351	;/BMGR-BUFFER MANAGER*/
		352	;
		353	DO;
		354	IF IBF=FULL THEN
		355	DO;
		356	IF TYPE=DATA THEN
		357	DO;
		358	MOTOR=ON;
		359	TEMP=INCPNT (INPNT);
		360	;
0100		361	ORG 100H
		362	;
0100	D647	363	BMGR: JNIBF BBPRT
0102	7636	365	JF1 BBCMD
0104	9ABF	366	ANL P2, #MOTON
0106	F8	367	MOV A, INPNT
0107	3470	368	CALL INCQPT

LOC	OBJ	SEQ	SOURCE STATEMENT
		369	IF TEMP<>OUT\$POINT THEN
		370	DO;
		371	IN\$POINT=TEMP;
		372	IF FILL\$MODE=ON THEN
		373	DO;
		374	TEMP=SPACE;
		375	ELSE DO;
		376	TEMP=INPUT\$BUFFER;
		377	INTERRUPT=OFF;
		378	END;
		379	IF TEMP=EOL THEN
		380	DO;
		381	FILL\$MODE=ON;
		382	TEMP=SPACE;
		383	END;
		384	IF TEMP=CONTROL\$CODE THEN TEMP='!';
		385	BUFFER(IN\$POINT)=TEMP AND 03FH;
		386	IF IN\$POINT=BUFFER\$MAX THEN
		387	DO;
		388	FILL\$MODE=OFF;
		389	OUT\$POINT=BUFFER\$MIN;
		390	END;
		391	END;
		392	
		393	
0109	D9	394	BBL1: XRL A,OUTPNT
010A	C647	395	JZ BBPRT
010C	D9	396	XRL A,OUTPNT
010D	A8	397	MOV INPNT,A
010E	0A	398	IN A,P2
010F	3216	399	JB1 FILL
0111	22	400	IN A,DBB
0112	9ADF	401	ANL P2,#NOT (EXREQ)
0114	2418	402	JMP BBL1A
0116	2320	403	MOV A,#SPACE
0118	D30A	404	BBL1A: XRL A,#EOL
011A	9620	405	JNZ BBL1B
011C	8A02	406	ORL P2,#FMODE
011E	2428	407	JMP BBL1C
0120	D30A	408	BBL1B: XRL A,#EOL
0122	D228	409	JB6 BBL1C
0124	B228	410	JB5 BBL1C
0126	2321	411	MOV A,#EXCLAIM
0128	533F	412	BBL1C: ANL A,#03FH
012A	A0	413	MOV @INPNT,A
012B	F8	414	MOV A,INPNT
012C	D33F	415	XRL A,#BMAX
012E	9647	416	JNZ BBPRT
0130	9AFD	417	ANL P2,#NOT FMODE
0132	B918	418	MOV OUTPNT,#BMIN
0134	2447	419	JMP BBPRT
		420	;
		421	;
		422	INTERRUPT=OFF;
		423	IF (PORT0 AND 3)=2 THEN INTENA=OFF;
		424	IF (PORT0 AND 3)=3 THEN INTENA=ON;
		425	END;
0136	22	426	BBCMD: IN A,DBB
0137	9ADF	427	ANL P2,#NOT (EXREQ)
0139	5303	428	ANL A,#3
013B	323F	429	JB1 BBL2
013D	2447	430	JMP BBPRT
013F	1245	431	BBL2: JB0 BBL3
0141	9AFE	432	ANL P2,#NOT INTENA
0143	2447	433	JMP BBPRT
0145	8A01	434	BBL3: ORL P2,#INTENA
		435	\$ EJECT

LOC	OBJ	SEQ	SOURCE STATEMENT
		436	;
		437	IF OBUF=EMPTY\$FLAG AND (OUT\$POINT<>BMIN OR STATE
		438	DO;
		439	IF OUT\$POINT<=BUFFER\$MAX THEN
		440	DO;
		441	OBUF=BIN(OUT\$POINT);
		442	OUT\$POINT=OUT\$POINT+1;
		443	END;
		444	END;
0147	FB	445	BBPRT: MOV A,OBUF
0148	D3FF	446	XRL A,#EMTFLG
014A	965E	447	JNZ BINT
014C	F9	448	MOV A,OUTPNT
014D	D318	449	XRL A,#OPTMIN
014F	9658	450	JNZ BBPRTA
0151	FE	451	MOV A,STATE
0152	03FD	452	ADD A,#-3
0154	F258	453	JB7 BBPRTA
0156	245E	454	JMP BINT
0158	F9	455	BBPRTA: MOV A,OUTPNT
0159	D25E	456	JB6 BINT
015B	F1	457	MOV A,OUTPNT
015C	AB	458	MOV OBUF,A
015D	19	459	INC OUTPNT
		460	;
		461	TEMP=INCQPT(INPNT);
		462	IF TEMP<>OUT\$POINT THEN
		463	DO;
		464	IF INTENA=ON AND FMODE=OFF THEN INTERRUPT=ON;
		465	END;
		466	END;
		467	END
015E	F8	468	BINT: MOV A,INPNT
015F	3470	469	CALL INCQPT
0161	D9	470	XRL A,OUTPNT
0162	C600	471	JZ BMGR
		472	BINTA:
0164	0A	473	IN A,P2
0165	37	474	CPL A
0166	1200	475	JB0 BMGR
0168	326C	476	JB1 SETINT
016A	2400	477	JMP BMGR
016C	8A20	478	SETINT: ORL P2,#EXREQ
016E	2400	479	JMP BMGR
		480	
		481	
		482	
		483	
		484	
		485	
		486	
		487	;
		488	PROCEDURE INCQPT(A,CARRY);
		489	DO;
		490	A=A MOD BUFFER LENGTH+BUFFER MIN;
		491	IF A=BUFFER_MIN THEN CARRY=1;
		492	END;
0170	0301	492	INCQPT: ADD A,#1
0172	D275	493	JB6 ADJUST
0174	83	494	RET
0175	2318	495	ADJUST: MOV A,#OPTMIN
0177	A7	496	CPL C
0178	83	497	RET
		498	
		499	
		500	;
		501	PROCEDURE INIT;
		502	DO;
		503	OBUF=EMPTY\$FLAG;
		504	OUT\$POINT=BUFFER\$MAX+1;
		505	IN\$POINT=BUFFER\$MAX;
		506	MOTOR=OFF;
		507	PAPER\$FEED=OFF;
		508	FILL\$MODE=OFF;
		509	INTERRUPT\$MASK=OFF;
		510	BUS\$BUFFER=0;
		511	END;
0179	BBFF	511	INIT: MOV OBUF,#EMTFLG
017B	B940	512	MOV OUTPNT,#BMAX+1
017D	B83F	513	MOV INPNT,#BMAX
017F	23F0	514	MOV A,#0F0H
0181	3A	515	OUTL P2,A
0182	22	516	IN A,DBB
0183	83	517	RET
		518	
		519	
		520	
		521	
		522	
		523	
		524	\$ EJECT

LOC	OBJ	SEQ	SOURCE STATEMENT
		525	*****
		526	;
		527	COLUMN IS CALLED WITH ICNT EOL TO THE CURRENT COLUMN NUMBER
		528	AND PBUFF EOL TO THE CHARACTER TO BE CONVERTED.
		529	COLUMN RETURNS THE APPROPRIATE COLUMN OF DATA FROM THE
		530	CHARACTER GENERATER TABLE. COLUMN IS LOCATED IN PAGE 2
		531	FOLLOWING THE FIRST HALF OF THE TABLE.
		532	;
		533	*****
		534	;
02E0		535	ORG 2E0H
		536	;
		537	PROCEDURE COLUMN (PRINT\$BUFFER, ICNT);
		538	DO;
		539	FLAG0=NOT PRINT\$BUFFER[5];
		540	PRINT\$BUFFER[5]=0;
		541	TEMP=7*(PBUF+1)-ICNT
		542	IF FLAG0=OFF THEN
		543	DO;
		544	TEMP=MP3 (TEMP);
		545	PRINT\$BUFFER[5]=1;
		546	END;
		547	ELSE DO;
		548	TEMP=MP2 (TEMP);
		549	END;
		550	END;
02E0	FC	551	COLUMN: MOV A, PBUF
02E1	85	552	CLR F0
02E2	B2E5	553	JB5 NOSET
02E4	95	554	CPL F0
02E5	531F	555	NOSET: ANL A, #01FH
02E7	AC	556	MOV PBUF, A
02E8	E7	557	RL A
02E9	E7	558	RL A
02EA	E7	559	RL A
02EB	37	560	CPL A
02EC	6C	561	ADD A, PBUF
02ED	6D	562	ADD A, ICNT
02EE	37	563	CPL A
02EF	0307	564	ADD A, #7
02F1	B6F9	565	JF0 PAG2
02F3	E3	566	MOVP3 A, @A
02F4	2C	567	XCH A, PBUF
02F5	4320	568	ORL A, #20H
02F7	2C	569	XCH A, PBUF
02F8	83	570	RET
02F9	A3	571	PAG2: MOVP A, @A
02FA	83	572	RET
		573	;
		574	;
		575	*****
		576	;
		577	*****
		578	;
		579	CHARACTER GENERATER TABLES.
		580	THE FIRST HALF OF THESE TABLES IS IN PAGE 2. FOLLOWING THIS HALF
		581	IS THE COLUMN SUBROUTINE. THE SECOND HALF OF THE TABLE IS
		582	IN PAGE 3. THE PLACEMENT OF THESE TABLES IS TO TAKE
		583	ADVANTAGE OF THE MCS-41 MOVP AND MOVP3 INSTRUCTIONS.
		584	;
		585	THE CHARACTERS ARE FORMED BY A SEVEN BY SEVEN MATRIX
		586	OF DOTS. EACH DOT POSITION CORRESPONDS
		587	TO ONE HALF THE NORMAL DOT SPACING OF THE LRC PRINTER.
		588	TO PREVENT EXCEEDING THE "BANDWIDTH" OF THE SOLENOIDS
		589	THE CHARACTERS ARE FORMED SO THAT THE SAME SOLENOID IS
		590	NOT ENERGIZED TWICE IN SUCCESSION. CONSTRUCTING THE
		591	TABLE IN THIS MANNER ALLOWS THE FORMATION OF A CHARACTER IN ONLY
		592	4 PRINT COLUMNS SINCE THREE OF THE DOTS WILL APPEAR BETWEEN
		593	NORMAL COLUMN POSITIONS.
		594	;
		595	THE COMMENT FIELD OF THE TABLE SHOWS THE BIT PATTERN OF THE
		596	CHARACTERS. THE SPACING OF THE PRINTER CHARACTERS CAUSES
		597	DISTORTION OF THE CHARACTERS BUT THE PATTERN IS STILL DISCERNABLE.
		598	;
		599	*****
		600	*****

LOC	OBJ	SEQ	SOURCE STATEMENT			
		601				
0200		602	ORG	200H		
		603				
		604				
0200	0C	605	DB	0CH	: **	; [AT]
0201	22	606	DB	22H	: * *	
0202	41	607	DB	41H	: * *	
0203	58	608	DB	58H	: * *	
0204	01	609	DB	01H	: *	
0205	48	610	DB	48H	: *	
0206	00	611	DB	00H	: *	
		612			:	
0207	0F	613	DB	0FH	: ****	; [A]
0208	10	614	DB	10H	: *	
0209	24	615	DB	24H	: *	
020A	40	616	DB	40H	: *	
020B	24	617	DB	24H	: *	
020C	10	618	DB	10H	: *	
020D	0F	619	DB	0FH	: ****	
		620			:	
		621			:	
020E	7F	622	DB	7FH	: *****	; [B]
020F	00	623	DB	00H	: *	
0210	49	624	DB	49H	: *	
0211	00	625	DB	00H	: *	
0212	08	626	DB	08H	: *	
0213	55	627	DB	55H	: *	
0214	22	628	DB	22H	: *	
		629			:	
0215	3E	630	DB	3EH	: *****	; [C]
0216	41	631	DB	41H	: *	
0217	00	632	DB	00H	: *	
0218	41	633	DB	41H	: *	
0219	00	634	DB	00H	: *	
021A	41	635	DB	41H	: *	
021B	22	636	DB	22H	: *	
		637			:	
021C	7F	638	DB	7FH	: *****	; [D]
021D	00	639	DB	00H	: *	
021E	41	640	DB	41H	: *	
021F	00	641	DB	00H	: *	
0220	00	642	DB	00H	: *	
0221	41	643	DB	41H	: *	
0222	3E	644	DB	3EH	: *****	
		645			:	
0223	7F	646	DB	7FH	: *****	; [E]
0224	00	647	DB	00H	: *	
0225	49	648	DB	49H	: *	
0226	00	649	DB	00H	: *	
0227	49	650	DB	49H	: *	
0228	00	651	DB	00H	: *	
0229	41	652	DB	41H	: *	
		653			:	
022A	7F	654	DB	7FH	: *****	; [F]
022B	00	655	DB	00H	: *	
022C	48	656	DB	48H	: *	
022D	00	657	DB	00H	: *	
022E	48	658	DB	48H	: *	
022F	00	659	DB	00H	: *	
0230	40	660	DB	40H	: *	
		661			:	
0231	3E	662	DB	3EH	: *****	; [G]
0232	41	663	DB	41H	: *	
0233	00	664	DB	00H	: *	
0234	41	665	DB	41H	: *	
0235	04	666	DB	04H	: *	
0236	41	667	DB	41H	: *	
0237	26	668	DB	26H	: ** *	
		669			:	
		670	EJECT			

023A 08	673	DB	08H	;	*	
023B 00	674	DB	00H	;		
023C 08	675	DB	08H	;	*	
023D 00	676	DB	00H	;		
023E 7F	677	DB	7FH	;	*****	
	678					
023F 00	679	DB	00H	;		[I]
0240 41	680	DB	41H	;	*	*
0241 00	681	DB	00H	;		
0242 7F	682	DB	7FH	;	*****	
0243 00	683	DB	00H	;		
0244 41	684	DB	41H	;	*	*
0245 00	685	DB	00H	;		
	686					
0246 02	687	DB	02H	;	*	[J]
0247 01	688	DB	01H	;	*	
0248 00	689	DB	00H	;		
0249 01	690	DB	01H	;	*	
024A 00	691	DB	00H	;		
024B 01	692	DB	01H	;	*	
024C 7E	693	DB	7EH	;	*****	
	694					
024D 7F	695	DB	7FH	;	*****	[K]
024E 00	696	DB	00H	;		
024F 04	697	DB	04H	;	*	
0250 14	698	DB	14H	;	*	*
0251 22	699	DB	22H	;	*	*
0252 41	700	DB	41H	;	*	*
0253 00	701	DB	00H	;		
	702					
0254 7F	703	DB	7FH	;	*****	[L]
0255 00	704	DB	00H	;		
0256 01	705	DB	01H	;	*	
0257 00	706	DB	00H	;		
0258 01	707	DB	01H	;	*	
0259 00	708	DB	00H	;		
025A 01	709	DB	01H	;	*	
	710					
025B 7F	711	DB	7FH	;	*****	[M]
025C 40	712	DB	40H	;	*	
025D 20	713	DB	20H	;	*	
025E 18	714	DB	18H	;	**	
025F 20	715	DB	20H	;	*	
0260 40	716	DB	40H	;	*	
0261 3F	717	DB	3FH	;	*****	
	718					
	719					
0262 7F	720	DB	7FH	;	*****	[N]
0263 20	721	DB	20H	;	*	
0264 10	722	DB	10H	;	*	
0265 08	723	DB	08H	;	*	
0266 04	724	DB	04H	;	*	
0267 00	725	DB	00H	;		
0268 7F	726	DB	7FH	;	*****	
	727					
0269 3E	728	DB	3EH	;	*****	[O]
026A 41	729	DB	41H	;	*	*
026B 00	730	DB	00H	;		
026C 41	731	DB	41H	;	*	*
026D 00	732	DB	00H	;		
026E 41	733	DB	41H	;	*	*
026F 3E	734	DB	3EH	;	*****	
	735					
0270 37	736	DB	37H	;	*** **	[P]
0271 00	737	DB	00H	;		
0272 48	738	DB	48H	;	*	*
0273 00	739	DB	00H	;		
0274 00	740	DB	00H	;		
0275 48	741	DB	48H	;	*	*
0276 30	742	DB	30H	;	**	
	743					
0277 3E	744	DB	3EH	;	*****	[Q]
0278 41	745	DB	41H	;	*	*
0279 00	746	DB	00H	;		
027A 40	747	DB	40H	;	*	*
027B 05	748	DB	05H	;	*	*
027C 42	749	DB	42H	;	*	*
027D 3D	750	DB	3DH	;	*****	

LOC	OBJ	SEQ	SOURCE STATEMENT
027E	7F	751	
027F	00	752	DB 7FH ***** ; [R]
0280	48	753	DB 00H
0281	00	754	DB 48H * *
0282	04	755	DB 00H
0283	4A	756	DB 04H * *
0284	31	757	DB 4AH * * *
		758	DB 31H * **
		759	
0285	32	760	DB 32H * ** ; [S]
0286	49	761	DB 49H * * *
0287	00	762	DB 00H
0288	49	763	DB 49H * * *
0289	00	764	DB 00H
028A	49	765	DB 49H * * *
028B	26	766	DB 26H * **
		767	
		768	
028C	40	769	DB 40H * ; [T]
028D	00	770	DB 00H
028E	40	771	DB 40H *
028F	3F	772	DB 3FH *****
0290	40	773	DB 40H *
0291	00	774	DB 00H
0292	40	775	DB 40H *
		776	
0293	7C	777	DB 7CH ***** ; [U]
0294	02	778	DB 02H *
0295	01	779	DB 01H *
0296	00	780	DB 00H
0297	01	781	DB 01H *
0298	02	782	DB 02H *
0299	7C	783	DB 7CH *****
		784	
029A	78	785	DB 78H ***** ; [V]
029B	04	786	DB 04H *
029C	02	787	DB 02H *
029D	01	788	DB 01H *
029E	02	789	DB 02H *
029F	04	790	DB 04H *
02A0	78	791	DB 78H *****
		792	
02A1	7E	793	DB 7EH ***** ; [W]
02A2	01	794	DB 01H *
02A3	02	795	DB 02H *
02A4	0C	796	DB 0CH **
02A5	02	797	DB 02H *
02A6	01	798	DB 01H *
02A7	7E	799	DB 7EH *****
		800	
02A8	41	801	DB 41H * * * ; [X]
02A9	22	802	DB 22H * * *
02AA	14	803	DB 14H * *
02AB	08	804	DB 08H *
02AC	14	805	DB 14H * * *
02AD	22	806	DB 22H * * *
02AE	41	807	DB 41H * * *
		808	
02AF	40	809	DB 40H * * * ; [Y]
02B0	20	810	DB 20H *
02B1	10	811	DB 10H *
02B2	0F	812	DB 0FH *****
02B3	10	813	DB 10H *
02B4	20	814	DB 20H *
02B5	40	815	DB 40H *
		816	
		817	\$ EJECT

LOC	OBJ	SEQ	SOURCE STATEMENT
02B6	41	818	DB 41H ; * *
02B7	02	819	DB 02H ; *
02B8	45	820	DB 45H ; * *
02B9	08	821	DB 08H ; *
02BA	51	822	DB 51H ; *
02BB	20	823	DB 20H ; *
02BC	41	824	DB 41H ; *
		825	
02BD	7F	826	DB 7FH ; ***** ; []
02BE	00	827	DB 00H ;
02BF	41	828	DB 41H ; *
02C0	00	829	DB 00H ;
02C1	41	830	DB 41H ; *
02C2	00	831	DB 00H ;
02C3	41	832	DB 41H ; *
		833	
02C4	40	834	DB 40H ; *
02C5	20	835	DB 20H ; *
02C6	10	836	DB 10H ; *
02C7	08	837	DB 08H ; *
02C8	04	838	DB 04H ; *
02C9	02	839	DB 02H ; *
02CA	01	840	DB 01H ; *
		841	
02CB	41	842	DB 41H ; *
02CC	00	843	DB 00H ; *
02CD	41	844	DB 41H ; *
02CE	00	845	DB 00H ; *
02CF	41	846	DB 41H ; *
02D0	00	847	DB 00H ; *****
02D1	7F	848	DB 7FH ;
		849	
02D2	00	850	DB 00H ; *
02D3	04	851	DB 04H ; *
02D4	08	852	DB 08H ; *
02D5	10	853	DB 10H ; *
02D6	08	854	DB 08H ; *
02D7	04	855	DB 04H ; *
02D8	00	856	DB 00H ; *
		857	
02D9	01	858	DB 01H ; *
02DA	00	859	DB 00H ; *
02DB	01	860	DB 01H ; *
02DC	00	861	DB 00H ; *
02DD	01	862	DB 01H ; *
02DE	00	863	DB 00H ; *
02DF	01	864	DB 01H ; *
		865	
		866	
		867	
		868	
		869	*****
		870	
		871	START OF SECOND HALF OF CGEN TABLE
		872	
		873	*****
		874	
0300		875	ORG 300H
		876	
		877	
0300	00	878	DB 00H ;
0301	00	879	DB 00H ;
0302	00	880	DB 00H ;
0303	00	881	DB 00H ;
0304	00	882	DB 00H ;
0305	00	883	DB 00H ;
0306	00	884	DB 00H ;
		885	
0307	00	886	DB 00H ;
0308	00	887	DB 00H ;
0309	00	888	DB 00H ;
030A	7D	889	DB 7DH ; * *****
030B	00	890	DB 00H ;
030C	00	891	DB 00H ;
030D	00	892	DB 00H ;
		893	
		894	\$ EJECT

LOC	OBJ	SEQ	SOURCE	STATEMENT
030E	00	895	DB	00H
030F	20	896	DB	20H
0310	40	897	DB	40H
0311	00	898	DB	00H
0312	20	899	DB	20H
0313	40	900	DB	40H
0314	00	901	DB	00H
		902		
0315	14	903	DB	14H
0316	00	904	DB	00H
0317	7F	905	DB	7FH
0318	00	906	DB	00H
0319	7F	907	DB	7FH
031A	00	908	DB	00H
031B	14	909	DB	14H
		910		
031C	00	911	DB	00H
031D	32	912	DB	32H
031E	49	913	DB	49H
031F	36	914	DB	36H
0320	49	915	DB	49H
0321	26	916	DB	26H
0322	00	917	DB	00H
		918		
0323	51	919	DB	51H
0324	02	920	DB	02H
0325	54	921	DB	54H
0326	08	922	DB	08H
0327	15	923	DB	15H
0328	20	924	DB	20H
0329	45	925	DB	45H
		926		
032A	26	927	DB	26H
032B	49	928	DB	49H
032C	10	929	DB	10H
032D	49	930	DB	49H
032E	26	931	DB	26H
032F	01	932	DB	01H
0330	05	933	DB	05H
		934		
0331	00	935	DB	00H
0332	00	936	DB	00H
0333	10	937	DB	10H
0334	20	938	DB	20H
0335	40	939	DB	40H
0336	00	940	DB	00H
0337	00	941	DB	00H
		942		
		943		
0338	1C	944	DB	1CH
0339	22	945	DB	22H
033A	41	946	DB	41H
033B	00	947	DB	00H
033C	00	948	DB	00H
033D	00	949	DB	00H
033E	00	950	DB	00H
		951		
033F	00	952	DB	00H
0340	00	953	DB	00H
0341	00	954	DB	00H
0342	00	955	DB	00H
0343	41	956	DB	41H
0344	22	957	DB	22H
0345	1C	958	DB	1CH
		959		
0346	49	960	DB	49H
0347	22	961	DB	22H
0348	1C	962	DB	1CH
0349	77	963	DB	77H
034A	1C	964	DB	1CH
034B	22	965	DB	22H
034C	49	966	DB	49H
		967		
034D	08	968	DB	08H
034E	08	969	DB	08H
034F	08	970	DB	08H
0350	3E	971	DB	3EH
0351	08	972	DB	08H
0352	08	973	DB	08H
0353	08	974	DB	08H

0355 00	977	DB	00H	; [,]
0356 00	978	DB	00H	
0357 01	979	DB	01H	*
0358 06	980	DB	06H	**
0359 00	981	DB	00H	
035A 00	982	DB	00H	
035B 04	983			
035C 04	984	DB	04H	*
035D 04	985	DB	04H	*
035E 04	986	DB	04H	*
035F 04	987	DB	04H	*
0360 04	988	DB	04H	*
0361 04	989	DB	04H	*
	990	DB	04H	*
	991			
	992			
0362 00	993	DB	00H	; [.]
0363 00	994	DB	00H	
0364 00	995	DB	00H	
0365 01	996	DB	01H	*
0366 00	997	DB	00H	
0367 00	998	DB	00H	
0368 00	999	DB	00H	
	1000			
0369 01	1001	DB	01H	*
036A 02	1002	DB	02H	*
036B 04	1003	DB	04H	*
036C 08	1004	DB	08H	*
036D 10	1005	DB	10H	*
036E 20	1006	DB	20H	*
036F 40	1007	DB	40H	*
	1008			
0370 1D	1009	DB	1DH	****
0371 22	1010	DB	22H	*
0372 45	1011	DB	45H	*
0373 08	1012	DB	08H	*
0374 51	1013	DB	51H	*
0375 22	1014	DB	22H	*
0376 5C	1015	DB	5CH	****
	1016			
0377 00	1017	DB	00H	; [1]
0378 21	1018	DB	21H	*
0379 40	1019	DB	40H	*
037A 7F	1020	DB	7FH	*****
037B 00	1021	DB	00H	*
037C 01	1022	DB	01H	*
037D 00	1023	DB	00H	*
	1024			
037E 23	1025	DB	23H	** *
037F 44	1026	DB	44H	*
0380 01	1027	DB	01H	*
0381 48	1028	DB	48H	*
0382 01	1029	DB	01H	*
0383 48	1030	DB	48H	*
0384 31	1031	DB	31H	*
	1032			
0385 42	1033	DB	42H	*
0386 01	1034	DB	01H	*
0387 50	1035	DB	50H	*
0388 01	1036	DB	01H	*
0389 50	1037	DB	50H	*
038A 29	1038	DB	29H	*
038B 46	1039	DB	46H	*
	1040			
	1041 \$	EJECT		

LOC	OBJ	SEQ	SOURCE STATEMENT
038C 04	1042	DB	04H ; * ; [4]
038D 08	1043	DB	08H ; * ;
038E 14	1044	DB	14H ; * * ;
038F 20	1045	DB	20H ; * * ;
0390 5F	1046	DB	5FH ; ***** ;
0391 00	1047	DB	00H ; * ;
0392 04	1048	DB	04H ; * ;
	1049		
0393 72	1050	DB	72H ; * *** ; [5]
0394 01	1051	DB	01H ; * * ;
0395 50	1052	DB	50H ; * * ;
0396 01	1053	DB	01H ; * * ;
0397 40	1054	DB	40H ; * * ;
0398 11	1055	DB	11H ; * * ;
0399 4E	1056	DB	4EH ; *** * ;
	1057		
039A 17	1058	DB	17H ; *** * ; [6]
039B 21	1059	DB	21H ; * * ;
039C 40	1060	DB	40H ; * * ;
039D 09	1061	DB	09H ; * * ;
039E 40	1062	DB	40H ; * * ;
039F 09	1063	DB	09H ; * * ;
03A0 46	1064	DB	46H ; * * ;
	1065		
03A1 40	1066	DB	40H ; * ; [7]
03A2 00	1067	DB	00H ; * ;
03A3 47	1068	DB	47H ; *** * ;
03A4 08	1069	DB	08H ; * * ;
03A5 50	1070	DB	50H ; * * ;
03A6 20	1071	DB	20H ; * ;
03A7 40	1072	DB	40H ; * ;
	1073		
03A8 36	1074	DB	36H ; ** ** ; [8]
03A9 49	1075	DB	49H ; * * * ;
03AA 00	1076	DB	00H ; * * * ;
03AB 49	1077	DB	49H ; * * * ;
03AC 00	1078	DB	00H ; * * * ;
03AD 49	1079	DB	49H ; * * * ;
03AE 36	1080	DB	36H ; ** ** ;
	1081		
03AF 30	1082	DB	30H ; ** ; [9]
03B0 48	1083	DB	48H ; * * * ;
03B1 01	1084	DB	01H ; * * * ;
03B2 48	1085	DB	48H ; * * * ;
03B3 01	1086	DB	01H ; * * * ;
03B4 42	1087	DB	42H ; * * * ;
03B5 3C	1088	DB	3CH ; ***** ;
	1089		
	1090		
03B6 00	1091	DB	00H ; ; [:]
03B7 00	1092	DB	00H ; ;
03B8 00	1093	DB	00H ; ;
03B9 14	1094	DB	14H ; * * ;
03BA 00	1095	DB	00H ; ;
03BB 00	1096	DB	00H ; ;
03BC 00	1097	DB	00H ; ;
	1098		
03BD 00	1099	DB	00H ; ; [;]
03BE 00	1100	DB	00H ; ;
03BF 01	1101	DB	01H ; * ;
03C0 02	1102	DB	02H ; * * ;
03C1 14	1103	DB	14H ; * * ;
03C2 00	1104	DB	00H ; ;
03C3 00	1105	DB	00H ; ;
	1106		
03C4 00	1107	DB	00H ; ; [<]
03C5 08	1108	DB	08H ; * * ;
03C6 14	1109	DB	14H ; * * * ;
03C7 22	1110	DB	22H ; * * * ;
03C8 41	1111	DB	41H ; * * * ;
03C9 00	1112	DB	00H ; ;
03CA 00	1113	DB	00H ; ;
	1114		
03CB 00	1115	DB	00H ; ; [=]
03CC 14	1116	DB	14H ; * * ;
03CD 00	1117	DB	00H ; ;
03CE 14	1118	DB	14H ; * * ;
03CF 00	1119	DB	00H ; ;
03D0 14	1120	DB	14H ; * * ;
03D1 00	1121	DB	00H ; ;

LOC	OBJ	SEQ	SOURCE STATEMENT
		1122	
03D2	00	1123	DB 00H ; [>]
03D3	00	1124	DB 00H ;
03D4	41	1125	DB 41H ; *
03D5	22	1126	DB 22H ; *
03D6	14	1127	DB 14H ; *
03D7	08	1128	DB 08H ; *
03D8	00	1129	DB 00H ;
		1130	
03D9	00	1131	DB 00H ; ; [?]
03DA	20	1132	DB 20H ; *
03DB	40	1133	DB 40H ; *
03DC	05	1134	DB 05H ; *
03DD	48	1135	DB 48H ; *
03DE	30	1136	DB 30H ; **
03DF	00	1137	DB 00H ;
		1138	
		1139	END

11-3	Contents
11-3	Introduction
11-3	The 8295
11-3	The LRC 7040 Printer
11-3	8295 Printer Interfaces
11-3	8295 Command Software
11-7	Parallel Interfaces
11-13	Serial Interfaces
11-15	8295 Software
11-16	Conclusion
11-17	Appendix A

April 1979

Using the 8295 Dot Matrix Printer Controller

John Beaton
Microcomputer Applications

Using the 8295 Dot Matrix Printer Controller

Contents

Introduction	11-3
The 8295	11-3
The LRC 7040 Printer	11-3
8295/Printer Interface	11-5
8295 Command Software	11-6
Parallel Interface	11-7
Serial Interface	11-13
8295 Software	11-15
Conclusion	11-16
Appendix A	11-17

INTRODUCTION

Many microprocessor systems require the real-time control of a peripheral device such as a printer, keyboard, or alpha-numeric display, etc. These medium speed but still real-time tasks can be rather mundane, time-consuming, and require a fair amount of system software overhead. Of course, any time spent by the main processor in servicing these I/O devices is unavailable for other, possibly more important, tasks. This processor burden can largely be removed by isolating the real-time portion of the task to a dedicated peripheral-control processor.

Until recently, this approach was usually not cost effective due to the large number of components required by the dedicated processor: CPU, RAM, ROM, I/O, etc. To help make the approach more cost effective, Intel borrowed the I/O processing concepts found in many mainframe and minicomputers; put all the hardware in one package; and introduced a family of Universal Peripheral Interface controllers—the UPI-41A™ family. The basic family consists of the 8041A and the 8741A. These two devices are essentially single-chip microcomputers with a standard microprocessor bus interface. They have on-chip RAM, ROM (8041A) or EPROM (8741A), CPU, timer/counter, and I/O. Using one of the UPI family, the designer simply codes his custom or proprietary peripheral control algorithm into the UPI device itself rather than the main system software. The UPI device then takes over the peripheral control task while the host processor simply issues commands and transfers data. More information on the UPI family is available in the documents referenced opposite the table of contents.

Illustrating the UPI concept as both design examples and actual products, a number of pre-programmed 8041As are available. These devices are the 8278 Keyboard/Display Controller, the 8294 Data Encryption Unit, the 8292 GPIB Controller, and the 8295 Dot Matrix Printer Controller. Data sheets for these devices are found in the Peripheral Design Handbook and their source listings (except for the 8294) are available in Insite, Intel's User's library. This application note deals with the 8295.

THE 8295

The 8295 Dot Matrix Printer Controller is a device specifically designed to interface microprocessors to the LRC 7040 Series of dot matrix impact printers. It offers complete solenoid and motor drive timing and contains an on-chip 7×7 character generator accommodating 64 ASCII characters. An on-chip FIFO buffers up to 40 ASCII characters before printing. Character density, width, and print intensity are all programmable. Three programmable tabulations and two general purpose outputs are also provided. Four data transfer methods are possible: polling, interrupt-driven, and Direct Memory

Access (DMA) are available when in parallel data transfer mode and asynchronous serial is available in serial mode. The data transfer mode is hardware selectable.

Let's first look at the LRC printer itself and its interface to the 8295.

THE LRC 7040 PRINTER

The LRC Model 7040 printer is manufactured by LRC, Inc. of Riverton, Wyoming. Capable of printing 40 columns of characters at a speed of 1.25 lines/sec, the 7040 is mechanically simple and is ideal for point-of-sale or data logging terminals.

It is an impact printer whose print head consists of seven solenoids which each drives a stiff wire to impact the paper through an inked ribbon. While the wires are arranged in a circular fashion at the solenoid end, they form a vertical column at the ribbon impact point. Characters are formed by firing the solenoids to form a 5×7 or 7×7 matrix of "dots" (impacts of the wires). Figure 1 shows how the character A is formed using a 7×7 matrix. The columns are labeled C1 thru C7 and the rows R1 thru R7. The print head moves left to right across the paper so at time T1, the head is over column C1. If the correct solenoids are activated at each time Tx for each column Cx, the character is formed.

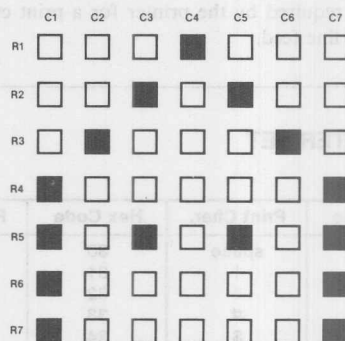


Figure 1. Character A in 7×7 Format

The print head is moved across the paper by the main motor drive. The main motor drive consists of a 24-pole synchronous motor which drives a rotating plastic drum. The drum has a spiral groove molded into it and a pin on the print head rests in the groove so that the print head traverses the paper as the drum rotates. Characters are printed by firing the solenoids during the left-to-right traverse. At the end of the print area, the spiral groove reverses the direction of the print head returning it to its home position.

A HOME microswitch riding on a cam attached to the plastic drum provides the only feedback as to the print head position. When the print head is in its home resting position the HOME switch is inactive. To start a print cycle, the main motor drive is activated which starts the print head motion. As the print head reaches the beginning of the print area, the cam activates the HOME switch as a signal to the printer controller to commence firing the solenoids. The controller then activates the solenoids as appropriate for each character in the line. The print area is defined as the 310ms immediately after HOME goes active. Solenoid timing is the responsibility of the controller; the printer mechanism supplies no character-position information.

After the line is printed and the print head has traversed right to left, the HOME switch is deactivated. This transition signals the controller to turn off the main motor drive since the home position has been reached. A new print cycle may start immediately if data is ready.

Paper feed is accomplished with a second synchronous motor and a PFEED (Paper Feed) microswitch. In the quiescent state, the PFEED switch is inactive. Activating the paper feed motor drive starts the line feed cycle. The switch becomes active at some point during the cycle (typically about 48ms later) and is deactivated when the cycle is complete. The controller uses the active-to-inactive transition to remove the paper feed motor drive. The paper feed operation is independent of the print cycle so the two could occur simultaneously. Figure 2 shows the timing required by the printer for a print cycle followed by a line feed.

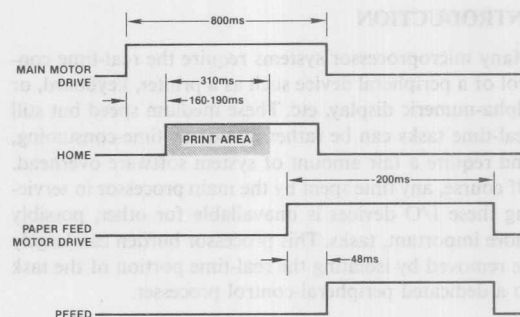


Figure 2. LRC 7040 Motor Drive Timing

Solenoid timing determines the location of any given "dot" and its intensity. The LRC 7040 printer specification states a 400 μ s maximum solenoid "ON" time and a 1.3ms typical period. Since the print area is 310ms "long," this timing allows a total of 240 dots (310ms/1.3ms per dot) in one row or 40 characters on a 5 \times 7 matrix with a one dot space between characters. While 5 \times 7 characters have acceptable readability, their distinctness and format can be improved with a 7 \times 7 matrix, however, 40 7 \times 7 characters translate to 320 dots per row or a 0.97ms solenoid period. This violates the solenoid duty cycle spec if the solenoids are fired for every column. The best way to get around this dilemma and still retain the improved readability of the 7 \times 7 format is to simply fire the solenoid every other column. The 8295 uses this technique and the "every-other" column spacing is reflected in Figure 1. The 8295 character set is included in Figure 3.

CHARACTER SET

Hex Code	Print Char.	Hex Code	Print Char.	Hex Code	Print Char.	Hex Code	Print Char.
20	space	30	0	40	@	50	P
21	!	31	1	41	A	51	Q
22	"	32	2	42	B	52	R
23	#	33	3	43	C	53	S
24	\$	34	4	44	D	54	T
25	%	35	5	45	E	55	U
26	&	36	6	46	F	56	V
27	,	37	7	47	G	57	W
28	(38	8	48	H	58	X
29)	39	9	49	I	59	Y
2A	.	3A	:	4A	J	5A	Z
2B	+	3B	;	4B	K	5B	[
2C	,	3C	<	4C	L	5C	\
2D	-	3D	=	4D	M	5D]
2E	.	3E	>	4E	N	5E	^
2F	/	4F	?	4F	O	5F	_

Figure 3. 8295 Character Set

8295/Printer Interface

It's the job of the 8295/Printer interface to convert the TTL-compatible outputs of the 8295 to the motor and solenoid drive levels. Since the printer side of the 8295 is independent of the system side, this same 8295/Printer interface is used for all examples discussed in the later sections.

For solenoid drive, the 8295 supplies seven solenoid outputs, S1 thru S7, plus a solenoid strobe, STB. STB modulates the S1-S7 outputs externally to supply the actual solenoid "ON" time. This time is software programmable. Figure 4 shows the recommended S1-S7/STB gating.

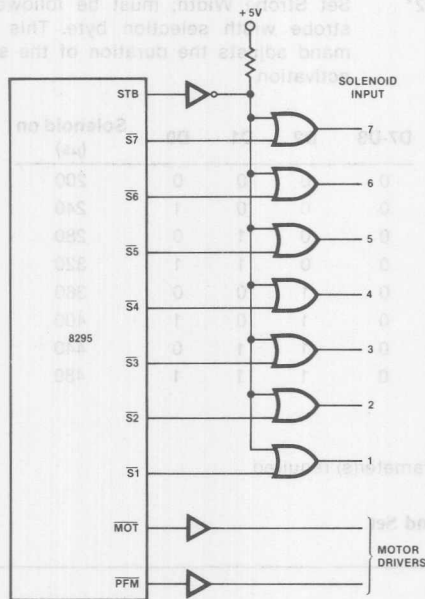


Figure 4. Solenoid and Motor Gating

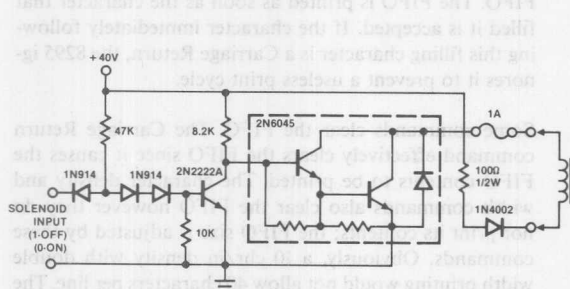


Figure 5. Solenoid Driver

The solenoids must be driven from a $40 \pm 10\%$ volt source. The peak current is approximately 3.6A, the average current is approximately 0.5A. A circuit providing the required drive is shown in Figure 5. The output stage, consisting of the 2N6045 Darlington transistor, the 1N4002 catching diode, and the 100-ohm damping resistor, is the one suggested by the manufacturer. The input stage is a discrete implementation of a DTL gate. Note that the base-emitter junction of the 2N6045 protects the 2N2222A transistor from overvoltage on its collector. This circuit has several features which are important to the printer interface:

1. All solenoid power (including the power used to drive the base of the power transistor) is derived from the 40-volt supply.
2. Disconnecting the drivers from the 8295 or the loss of the 5-volt supply to the 8295 results in the solenoids turning off.

The first feature of the drivers minimizes the impact of the printer and its interface on the 5-volt supply. The second feature prevents the activation of the solenoids erroneously during power on/off cycles or during system checkout. This is an important point since the solenoids will be damaged if left activated continuously. The fuses in series with the solenoids help protect them from mishap.

The two motors can each be driven as shown in Figure 6. The Monsanto MCS-6200 is an optically-coupled TRIAC which is ideal for driving the small synchronous motors in the printer. Coupled with a buffer this part provides a simple means of controlling the motors without sacrificing the isolation required for safe and reliable operation.

These driver circuits were borrowed from the Intel application note AP-27 "Printer Control With the UPI-41." (The 8295 development was inspired by the success of the AP-27 design.) Other solenoid and motor driver circuits are described in the LRC Interface Guide available from the manufacturer.

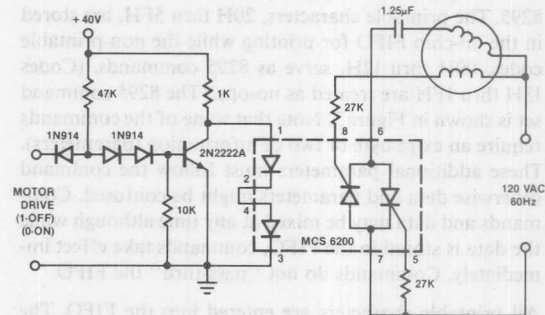


Figure 6. Motor Driver

- pin to a logic low state. After power on it is automatically set high.
- 01 Clear GP2. Same as the above but for GP2.
 - 02 Set GP1. Sets GP1 pin to a logic high state, inverse of command 00.
 - 03 Set GP2. Same as above but for GP2. Inverse command 01.
 - 04 Software Reset. This is a pacify command. This command is not effective immediately after commands requiring a parameter, as the Reset command will be interpreted as a parameter.
 - 05 Print 10 characters/in. density.
 - 06 Print 12 characters/in. density.
 - 07 Print double width characters. This command prints characters at twice the normal width, that is, at either 17 or 20 characters per line.
 - 08* Enable DMA mode; must be followed by two bytes specifying the number of data characters to be fetched. Least significant byte accepted first.
 - 09 Tab character.
 - 0A Line feed.
 - 0B* Multiple Line Feed; must be followed by a byte specifying the number of line feeds.
 - 0C Top of Form. Enables the line feed output until the Top of Form input is activated.

- 0E* enables the printer to start printing.
- 0F* Set Tab #1, followed by tab position byte. Should be greater than Tab #1.
- 10* Set Tab #3, followed by tab position byte. Should be greater than Tab #1.
- 11 Print Head Home on Right. On some printers the print head home position is on the right. This command would enable normal left to right printing with such printers.
- 12* Set Strobe Width; must be followed by strobe width selection byte. This command adjusts the duration of the strobe activation.

D7-D3	D2	D1	D0	Solenoid on (μs)
0	0	0	0	200
0	0	0	1	240
0	0	1	0	280
0	0	1	1	320
0	1	0	0	360
0	1	0	1	400
0	1	1	0	440
0	1	1	1	480

* parameter(s) required

Figure 7. 8295 Command Set

8295 Command Software

The software control of the 8295 is very straightforward. The host processor simply issues ASCII characters to the 8295. The printable characters, 20H thru 5FH, are stored in the on-chip FIFO for printing while the non-printable codes, 00H thru 12H, serve as 8295 commands. (Codes 13H thru 1FH are treated as no-ops.) The 8295 command set is shown in Figure 7. Note that some of the commands require an extra byte or two of information (parameters). These additional parameters must follow the command otherwise data and parameters might be confused. Commands and data may be mixed at any time although while the data is stored in the FIFO, commands take effect immediately. Commands do not "pass-thru" the FIFO.

All printable characters are entered into the FIFO. The FIFO is printed when either a Carriage Return command is received or the FIFO becomes full. In either case, the FIFO is printed, however there is no automatic line feed

unless the printer happens to be so equipped mechanically. Thus, a Line Feed command should be issued after each Carriage Return or after the last character to fill the FIFO. The FIFO is printed as soon as the character that filled it is accepted. If the character immediately following this filling character is a Carriage Return, the 8295 ignores it to prevent a useless print cycle.

Some commands clear the FIFO. The Carriage Return command effectively clears the FIFO since it causes the FIFO contents to be printed. The character density and width commands also clear the FIFO however they do not print its contents; the FIFO size is adjusted by these commands. Obviously, a 10 chr/in density with double width printing would not allow 40 characters per line. The 8295 recognizes this fact and modifies internally the FIFO size limits. The FIFO size is modified according to the table below. For example, if the density is 10 char/in, single width printing, the 8295 accepts only 33 printable

characters before starting a print cycle. Since these commands take effect as soon as they are accepted, this prevents mixing different character densities or widths on a given line. Any such commands must precede the data for a line.

DENSITY	WIDTH	BUFFER SIZE
12	SINGLE	40
12	DOUBLE	20
10	SINGLE	33
10	DOUBLE	17

The Software Reset command clears the FIFO, resets the density to 12 chr/in and selects single width printing. It does not effect the solenoid strobe width, the tab positions, or the general purpose outputs. This command should be issued only when the 8295 is expecting a command or data. Issuing it when the 8295 is expecting a parameter causes it to be interpreted as the parameter and not the intended software reset.

A hardware reset causes the 8295 to default into the following states:

1. Clears the FIFO
2. GP1 and GP2 set high
3. 12 chr/in density
4. single width printing
5. 320 μ s strobe width
6. tab positions indeterminate.

Parallel Interfaces

The 8295 has the option of using serial or parallel communication with the main processor. The choice must be

made early in the design cycle since it is a hardware, not a software, selection. Let's look at the parallel options first.

In parallel mode, the 8295 has the traditional microprocessor bus interface: data, control, etc. The parallel mode is selected by not grounding the IRQ/SER pin. To the main processor, the 8295 in parallel mode appears as two registers: the Input Data register and the Output Status register. The main processor writes commands and data into the Input Data register while it reads the 8295 status from the Output Status register.

The Output Status register format is shown in Figure 8. The Input Buffer Full bit (IBF) indicates whether the 8295 has accepted the previous command or data byte. IBF is automatically set when the host processor writes to the 8295 and it is reset when the 8295 accepts the data or command. If IBF = 1, no writes to the Input Data register are allowed. Only when IBF = 0 may a Input Data register write be done. The DMA Enable bit (DE) is set whenever the 8295 is performing DMA data transfers. When the specified number of transfers has been made, the DE bit is cleared. Since DMA cycles are usually transparent to the main processor, the DE bit tells the processor when the DMA block transfer is complete.

The processor does not always have to read the Output Status register, checking IBF, before loading the Input Data register. An interrupt output (IRQ) pin is available to interrupt the processor whenever the 8295 is ready to receive new data or commands. The fact that IRQ is set implies that IBF = 0, so it's not necessary for the processor to read the 8295 status when interrupted; it can just write the next byte.

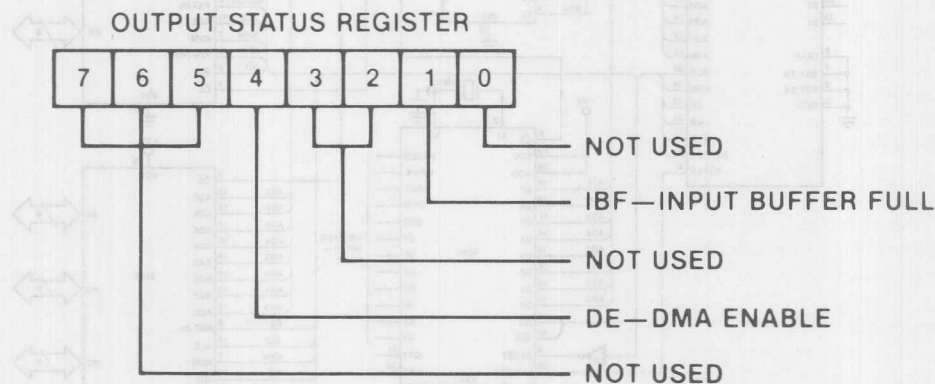


Figure 8. Output Status Register Format

Figure 9 shows the system schematic for using the 8295 in polled-parallel mode in an 8085A system; ie the IRQ line is not used. The 8085A/8295 interface is standard as for any Intel peripheral. \overline{CS} is decoded from the high-order address lines. \overline{RD} and \overline{WR} are the 8085A read and write control lines. \overline{RESET} is the system reset.

Example 8085A polling software is shown in Figure 10. This routine simply outputs the print buffer starting at the location pointed to in PRTSRT. The system software builds the buffer, terminates it with a 0FFH character, and loads PRTSRT before calling PRINT.

PRINT is not very efficient with respect to processing time. Since the 8295 does not accept data while in a print or line feed cycle, if the buffer contained more printable characters than the FIFO size, the processor would sit in the PRT2 loop during the 800ms print and 200ms line feed cycles. That is obviously not too efficient. The obvious way around this problem is to restrict the buffer size to less than that of the FIFO however this could complicate the system software since more buffer building is required. A better approach is to use interrupts.

By connecting the 8295's IRQ output to one of the 8085A RST interrupt inputs (dotted line in Figure 9), the pro-

cessor is interrupted only when the 8295 is able to take another character. Figure 11 shows such interrupt-driven software assuming the RST 6.5 interrupt input is used for IRQ.

To further enhance the bus efficiency and processor overhead at the expense of slightly more complex hardware, use the 8295 DMA interface. This DMA interface is compatible with the 8257 DMA Controller. With such an interface all that's necessary is for the processor to load the DMA Controller with the print buffer starting address and write the Enable DMA command and length parameters into the 8295. The 8295 does the rest by requesting data directly from memory thru the DMA Controller. It keeps track of the number of characters to request. As long as there are characters remaining to be transferred, the DE bit in the Output Status register is set. After the last byte is transferred into the 8295, the DE bit is reset and the IRQ is made active. Either event is used to tell the processor that DMA is complete and the 8295 is ready for the next block. It is not necessary to restrict the DMA block size to 40 characters, the Enable DMA command parameters allow for up to 65k byte block sizes. The block size given the 8295 must reflect both data plus commands and parameters.

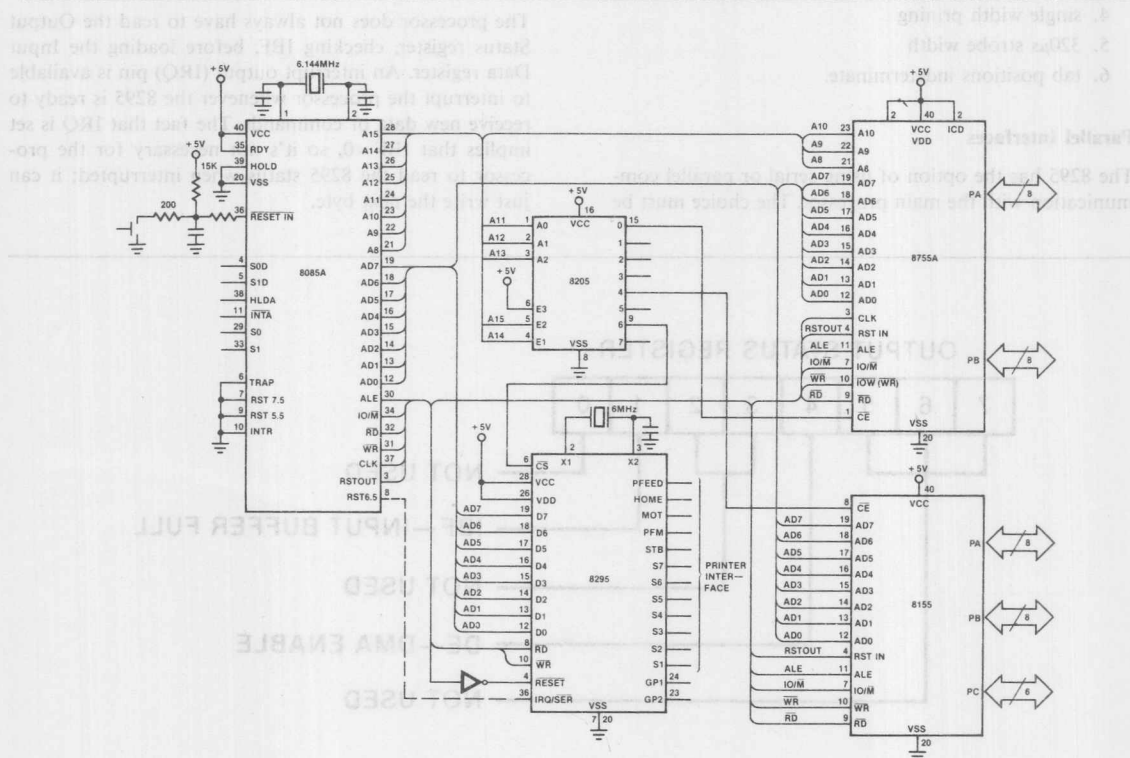


Figure 9. 8295 Parallel Interface

ASM80 :F1:95F10 SRC TITLE('8295 AP NOTE FIGURE 10')

ISIS-II 8080/8085 MACRO ASSEMBLER, X108 MODULE
8295 AP NOTE FIGURE 10

LOC	OBJ	SEQ	SOURCE STATEMENT				
		1	#MOD85				
		2	;				
		3	:SYSTEM EQUATES				
2000		4	PR1SRT EQU 2000H ; POINTER STORAGE				
0002		5	IBF EQU 02H ; IBF FLAG MASK				
0031		6	STS95 EQU 31H ; 8295 STATUS REGISTER PORT				
0031		7	DATA95 EQU 31H ; 8295 DATA REGISTER PORT				
		8	;				
2030		9	ORG 2030H				
		10	;				
		11	:PRINT BUFFER OUTPUT SUBROUTINE - THIS ROUTINE PRINTS THE BUFFER				
		12	:STARTING AT THE POINTER STORED AT PR1SRT. THE ROUTINE RETURNS WHEN				
		13	:A 0FFH IS FETCHED FROM THE BUFFER.				
		14	;				
2030 E5		15	PRINT: PUSH H ; SAVE HL				
2031 C5		16	PUSH B ; SAVE BC				
2032 2FD020		17	LHLD PR1SRT ; GET BUFFER POINTER				
2035 7E		18	PRT1: MOV A,M ; GET CHARACTER FROM BUFFER				
2036 47		19	MOV B,A ; SAVE I1 IN B				
2037 FFFF		20	CPI 0FFH ; IS IT THE BUFFER END?				
2039 CA4A20		21	JZ PEXIT ; YES, GO EXIT				
203C DB31		22	PRT2: IN STS95 ; NO, READ 8295 STATUS				
203E E602		23	ANI IBF ; LOOK AT IBF FLAG				
2040 C23C20		24	JNZ PRT2 ; WAIT UNTIL IBF=0				
2043 78		25	MOV A,B ; RECOVER CHARACTER				
2044 D331		26	OUT DATA95 ; OUTPUT TO 8295				
2046 23		27	INX H ; BUMP BUFFER POINTER				
2047 C33520		28	JMP PRT1 ; GET NEXT CHARACTER				
		29	;				
204A C1		30	PEXIT: POP B ; RESTORE BC				
204B E1		31	POP H ; RESTORE HL				
204C C9		32	RET ; RETURN				
		33	;				
		34	END				
PUBLIC SYMBOLS							
EXTERNAL SYMBOLS							
USER SYMBOLS							
DATA95 A 0031	IBF A 0002	PEXIT A 204A	PRINT A 2030	PRT1 A 2035	PRT2 A 203C	PR1SRT A 2000	STS95 A 0031

ASSEMBLY COMPLETE, NO ERRORS

Figure 10. 8085A/8295 Polling Subroutine

LOC	OBJ	SEQ	SOURCE STATEMENT	TERMINAL STATEMENT	LOC	OBJ
		1	\$MOD85			
		2	;			
		3	;SYSTEM EQUATES			
2000		4	PRTSRT EQU 2000H ; POINTER STORAGE			
0002		5	IBF EQU 02H ; IBF FLAG MASK			
0031		6	ST595 EQU 31H ; 8295 STATUS REGISTER PORT			
0031		7	DATA95 EQU 31H ; 8295 DATA REGISTER PORT			
		8	;			
		9	;			
		10	;RST6.5 INTERRUPT VECTOR LOCATION - JUMP TO PRINTER SUBROUTINE			
		11	;			
0034		12	ORG 3400H ; THIS ROUTINE - PRINT BUFFER STATUS			
0034	C33020	13	RST65: JMP PRINT ; GO TO PRINT ROUTINE			
		14	;			
		15	;			
2030		16	;			
		17	ORG 2030H			
		18	;			
		19	;PRINTER OUTPUT SUBROUTINE FOR INTERRUPT-DRIVEN SYSTEM - OUTPUTS			
		20	;CHR POINTED AT BY PRTSRT. IF CHR IS 0FFH, THE BUFFER IS COMPLETE			
		21	;AND THE RST6.5 INTERRUPT IS MASKED. THE MAIN PROGRAM MUST UNMASK			
		22	;RST6.5 AFTER IT BUILDS A NEW BUFFER. PRINT BUFFER STATUS IS REFLECTED			
		23	;TO THE MAIN PROGRAM BY THE RST6.5 MASK BIT IN RIM INSTRUCTION.			
		24	;			
2030	E5	25	PRINT: PUSH H ;SAVE HL			
2031	F5	26	PUSH PSW ;SAVE PSW			
2032	2A0020	27	LHLD PRTSRT ;GET BUFFER POINTER			
2035	7E	28	MOV A,M ;GET NEXT CHR			
2036	FEFF	29	CPI 0FFH ;TEST IF BUFFER COMPLETE			
2038	CA4520	30	JZ EXIT ;YES, GO EXIT WITH RST MASKED			
2038	D331	31	OUT DATA95 ;NO, OUTPUT CHR TO 8295			
203D	23	32	INX H ;BUMP POINTER			
203E	22D020	33	SHLD PRTSRT ;RESTORE POINTER			
2041	F1	34	POP PSW ;RESTORE PSW			
2042	E1	35	POP H ;RESTORE HL			
2043	FB	36	EI ;RE-ENABLE INTERRUPTS			
2044	C9	37	RET ;RETURN			
		38	;			
2045	3E0A	39	EXIT: MVI A,0AH ;MASK RST6.5			
2047	30	40	SIM ;SET INTERRUPT MASK			
2048	C34120	41	JMP PRT1 ;GO EXIT WITH MASK IN PLACE			
		42	;			
		43	END			
PUBLIC SYMBOLS						
EXTERNAL SYMBOLS						
USER SYMBOLS						
DATA95	A 0031	EX11	A 2045	IBF	A 0002	PRINT
				A 2030	PRT1	A 2041
				PRTSRT	A 2000	RST65
						A 0034

Figure 11. 8085A/8295 Interrupt-Driven Software

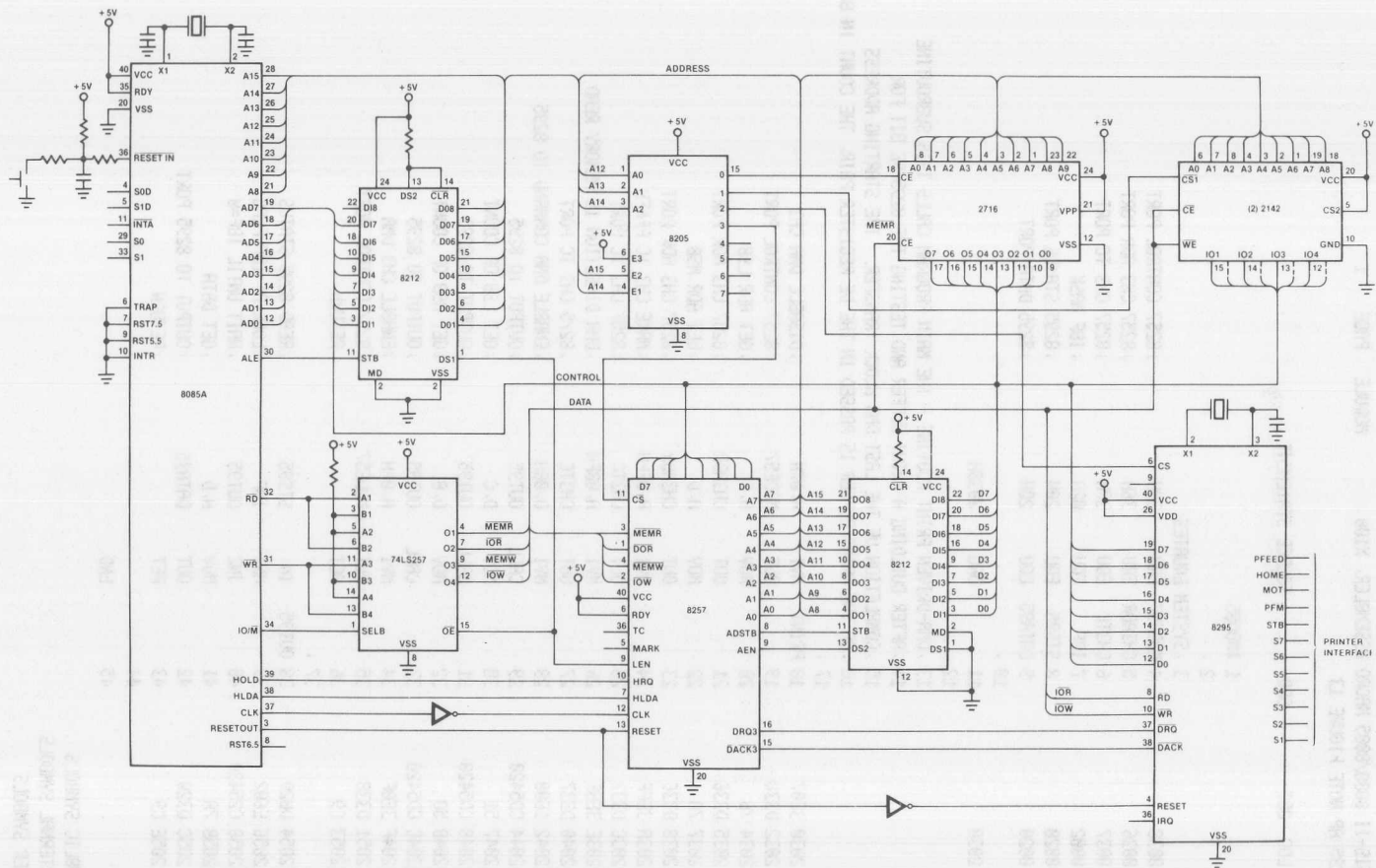


Figure 12. 8295/DMA Interface

LOC	OBJ	SEQ	SOURCE STATEMENT
		1	\$MOD85
		2	;
		3	;SYSTEM EQUATES:
0038		4	MODE57 EQU 38H ;8257 CONTROL PORT
0036		5	CH3ADR EQU 36H ;8257 CH3 ADR PORT
0037		6	CH3TC EQU 37H ;8257 CH3 TC PORT
0002		7	IBF EQU 02H ;IBF MASK
0020		8	STS95 EQU 20H ;8295 STATUS PORT
0020		9	DATA95 EQU 20H ;8295 DATA PORT
		10	;
2030		11	ORG 2030H
		12	;
		13	;DMA-DRIVEN PRINT ROUTINE - THE MAIN PROGRAM CALLS THIS SUBROUTINE
		14	;AFTER BUILDING A PRINT BUFFER AND TESTING THE 8295 DE BIT FOR
		15	;COMPLETION OF THE LAST DMA BLOCK TRANSFER. THE STARTING ADDRESS
		16	;OF THE PRINT BUFFER IS PASSED IN THE DE REGISTER PAIR, THE COUNT IN BC.
		17	;
2030 3E07		18	PRINT: MVI A,07H ;DISABLE DMA CH3
2032 D338		19	OUT MODE57 ;8257 CONTROL PORT
2034 7B		20	MOV A,E ;GET ADR LSB
2035 D336		21	OUT CH3ADR ;8257 CH3 ADR PORT
2037 7A		22	MOV A,D ;GET ADR MSB
2038 D33C		23	OUT CH3ADR ;8257 CH3 ADR PORT
203A 3EFF		24	MVI A,0FFH ;MAKE CH3 TC FFFFH
203C D337		25	OUT CH3TC ;8257 CH3 TC PORT
203E 3EBF		26	MVI A,0BFH ;DMA DIRECTION IS MEMORY READ
2040 D337		27	OUT CH3TC ;8257 CH3 TC PORT
2042 1608		28	MVI D,08H ;ENABLE DMA COMMAND TO 8295
2044 CD5420		29	CALL OUT95 ;OUTPUT TO 8295
2047 51		30	MOV D,C ;GET LSB OF COUNT
2048 CD5420		31	CALL OUT95 ;OUTPUT TO 8295
204B 50		32	MOV D,B ;GET MSB OF COUNT
204C CD5420		33	CALL OUT95 ;OUTPUT TO 8295
204F 3E0F		34	MVI A,0FH ;ENABLE CH3 DMA
2051 D338		35	OUT MODE57 ;8257 CONTROL PORT
2053 C9		36	RET ;RETURN
		37	;
2054 D820		38	OUT95 IN STS95 ;READ 8295 STATUS
2056 E602		39	ANI IBF ;LOOK AT IBF FLAG
2058 C25420		40	JNZ OUT95 ;WAIT UNTIL IBF=0
205B 7A		41	MOV A,D ;GET DATA
205C D320		42	OUT DATA95 ;OUTPUT TO 8295 PORT
205E C9		43	RET ;RETURN
		44	;
		45	END

PUBLIC SYMBOLS
EXTERNAL SYMBOLS
USER SYMBOLS

Figure 13. 8295 DMA Subroutine

Figure 12 illustrates an 8257/8295 interface and Figure 13 shows example software for handling the system. This software assumes that the 8295 is doing the counting of the transfers hence the Terminal Count of the 8257 DMA channel is loaded with the maximum value while the 8295 receives the actual block size. The 8295 simply stops making requests once the requested number of transfers have been made.

Serial Interface

In addition to the parallel interface options, the 8295 supports a "stand-alone" serial interface. In this mode, the only communication with the main processor is via a serial link. This configuration is perfect for remote printer applications; only three wires are required compared to 12 or 13 for the parallel interfaces.

The serial mode is invoked by simply grounding the IRQ/SER pin. See Figure 14. The internal 8295 software interrogates this pin upon power-on and reconfigures the function of several pins if it's grounded. The DACK/SIN pin becomes the serial data input (SIN) and the DRQ/CTS pin becomes the hardware data holdoff, Clear-to-Send. The lower three Data Bus pins become the Baud Rate Select inputs. Note that it is necessary to ground CS and WR, and pull RD high. This enables the "input" direction of the Data Bus pins so that the 8295 may read the baud rate. All standard baud rates from 110 to 4800 baud are accommodated.

After power-on the 8295 looks at IRQ/SER and if it's grounded, the data bus pins are read to determine the baud rate. Data from the serial input is requested by lowering CTS. CTS stays low until during the eight bit of the serial data character at which point it goes high (inactive). After the character is assembled and interpreted, CTS again goes active to request the next character. The 8295 does not check for parity and characters with invalid start bits or framing errors (stop bit wrong polarity) are ignored. CTS is normally connected to the UART's CTS input. An inactive CTS holds off the UART transmitter from transmitting characters.

In serial mode, the command and data definitions still apply as in parallel mode. Commands and data may be mixed although commands take effect immediately when received.

Figure 15 shows example software to drive an 8251A Programmable Serial Interface when connected to an 8295. This software is similar to Figure 10 except it assumes that the 8251A has the same I/O port addresses as the 8295 had in Figure 9. Note that the TXE (Transmitter Empty) flag is used to load data into the 8251A transmitting both characters in the transmitter (the transmitter is double buffered) if CTS goes inactive. The TXE flag allows only one character at a time in the transmitter so CTS going inactive simply finishes off the current character. The 8295 accepts only one character at a time.

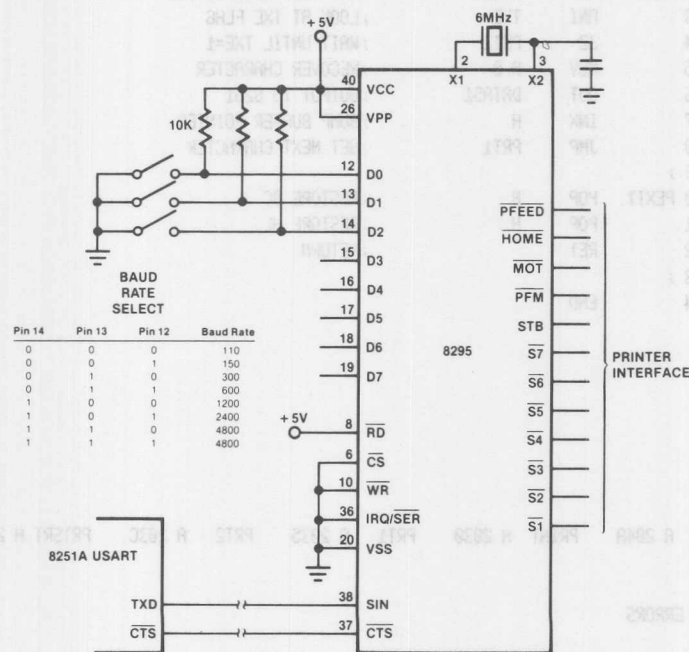


Figure 14. 8295 Serial Interface

ASM00 :F1:95F15.SRC TITLE('8295 AP NOTE FIGURE 15')

ISIS-II 8080/8085 MACRO ASSEMBLER, X108 MODULE PAGE 1
8295 AP NOTE FIGURE 15

LOC OBJ SEQ SOURCE STATEMENT

```

1 $MOD65
2 ;
3 ;SYSTEM EQUATES
4 PRTSRT EQU 2000H ; POINTER STORAGE
5 TXE EQU 04H ; TXE FLAG MASK
6 STS51 EQU 31H ; 8251 STATUS REGISTER PORT
7 DATA51 EQU 31H ; 8251 DATA REGISTER PORT
8 ;
9 ORG 2030H
10 ;
11 ;PRINT BUFFER OUTPUT SUBROUTINE - THIS ROUTINE PRINTS THE BUFFER
12 ;STARTING AT THE POINTER STORED AT PRTSRT. THE ROUTINE RETURNS WHEN
13 ;A 0FFH IS FETCHED FROM THE BUFFER
14 ;
15 PRINT: PUSH H ; SAVE HL
16 PUSH B ; SAVE BC
17 LHL PRTSRT ; GET BUFFER POINTER
18 PRT1: MOV A,M ; GET CHARACTER FROM BUFFER
19 MOV B,A ; SAVE IT IN B
20 CPI 0FFH ; IS IT THE BUFFER END?
21 JZ PEXIT ; YES, GO EXIT
22 PRT2: IN STS51 ; NO, READ 8251 STATUS
23 ANI TXE ; LOOK AT TXE FLAG
24 JZ PRT2 ; WAIT UNTIL TXE=1
25 MOV A,B ; RECOVER CHARACTER
26 OUT DATA51 ; OUTPUT TO 8251
27 INX H ; BUMP BUFFER POINTER
28 JMP PRT1 ; GET NEXT CHARACTER
29 ;
30 PEXIT: POP B ; RESTORE BC
31 POP H ; RESTORE HL
32 RET ; RETURN
33 ;
34 END

```

PUBLIC SYMBOLS

EXTERNAL SYMBOLS

USER SYMBOLS

DATA51 A 0031 PEXIT A 204A PRINT A 2030 PRT1 A 2035 PRT2 A 203C PRTSRT H 2000 STS51 A 0031
TXE A 0004

ASSEMBLY COMPLETE, NO ERRORS

Figure 15. 8251A Subroutine

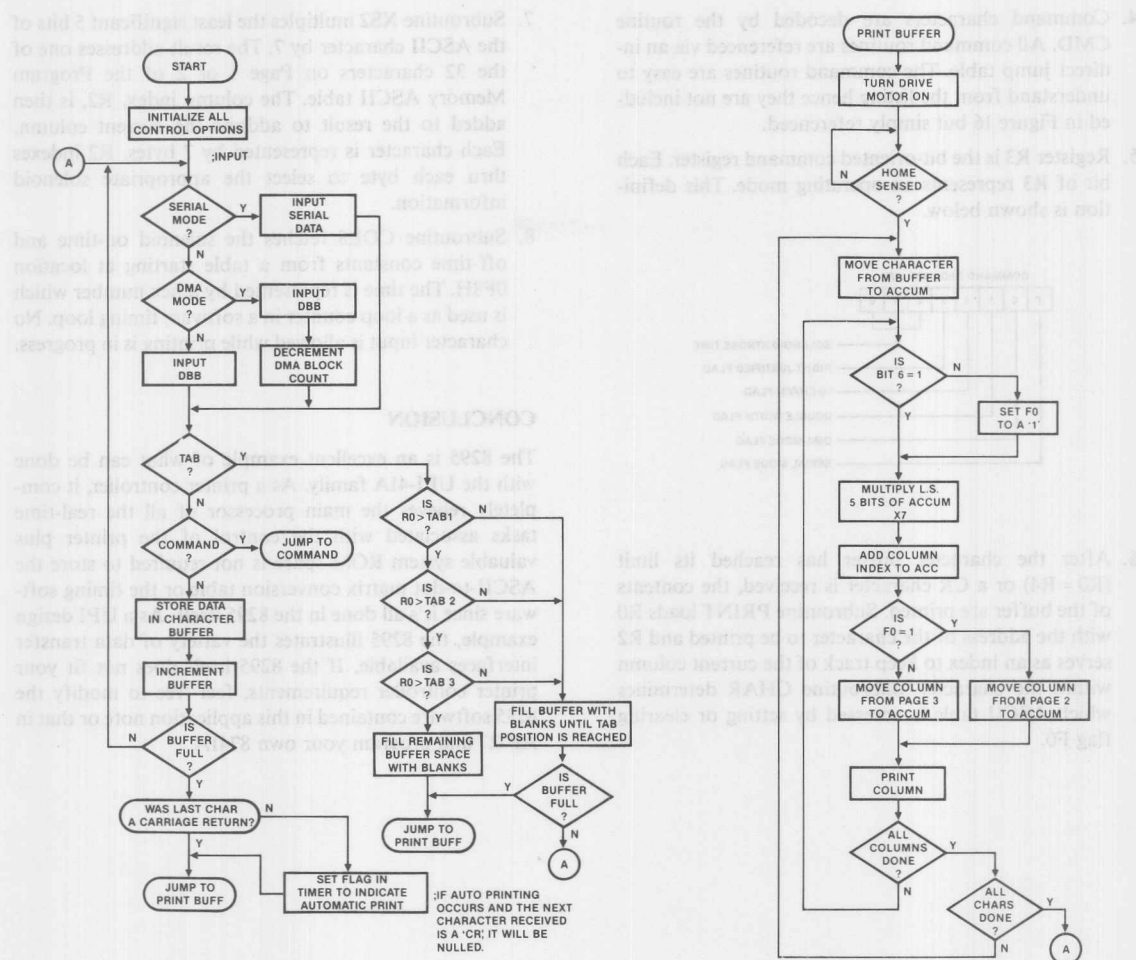


Figure 16. 8295 Flow Chart

8295 SOFTWARE

For those readers using the 8295 as a design example for UPI software, the flow charts for the program are shown in Figure 16 and the 8295 source listing is included as Appendix A. (Machine readable source listings are available through Insite, the Intel User's Library.) As an aid to understanding this software, the following observations can be made:

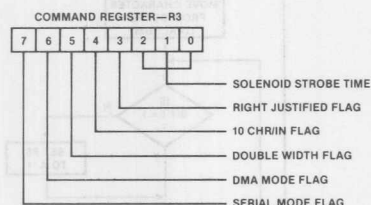
1. The 8295 uses only Register Bank 0. The function of registers R6 and R7 is determined by the mode. In parallel mode they are concatenated to form the 16 bit DMA count register. In serial mode, R6 is a counter during character reception.

2. Characters and commands are input from the Input Data register via the INPUT subroutine. The routine defines the input mode, fetches the data, and stores it in R2. If the DMA mode is enabled, the block count in R6 and R7 is decremented by the DECR routine each time a data transfer occurs until the count is exhausted.

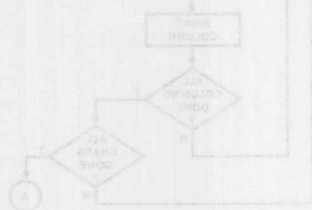
3. Characters are decoded by routine P6A which also detects any illegal characters by the INPUT routine. R0 is assigned as the character buffer pointer and R4 is designated as the buffer size limit. The commands which affect the buffer size will affect R0 and R4.

direct jump table. The command routines are easy to understand from the listing hence they are not included in Figure 16 but simply referenced.

5. Register R3 is the bit-oriented command register. Each bit of R3 represents an operating mode. This definition is shown below.



6. After the character buffer has reached its limit (R0=R4) or a CR character is received, the contents of the buffer are printed. Subroutine PRINT loads R0 with the address of the character to be printed and R2 serves as an index to keep track of the current column within the character. Subroutine CHAR determines which ASCII table is accessed by setting or clearing flag F0.



the 32 characters on Page 1 or 2 of the Program Memory ASCII table. The column index, R2, is then added to the result to address the current column. Each character is represented by 7 bytes. R2 indexes thru each byte to select the appropriate solenoid information.

8. Subroutine COL8 fetches the solenoid on-time and off-time constants from a table starting at location 0F8H. The time is represented by a hex number which is used as a loop counter in a software timing loop. No character input is allowed while printing is in progress.

CONCLUSION

The 8295 is an excellent example of what can be done with the UPI-41A family. As a printer controller, it completely relieves the main processor of all the real-time tasks associated with the control of the printer plus valuable system ROM space is not required to store the ASCII-to-dot matrix conversion table or the timing software since it's all done in the 8295 itself. As a UPI design example, the 8295 illustrates the variety of data transfer interfaces available. If the 8295 itself does not fit your printer controller requirements, feel free to modify the 8295 software contained in this application note or that in AP-27 and program your own 8741A.

Figure 16. 8295 Flow Chart

2. Characters and commands are input from the input register via the INPUT subroutine. The routine defines the input mode, fetches the data, and stores it in R1. If the DMA mode is enabled the block count in R0 and R1 is determined by the DECR routine each time a data transfer occurs until the count is exhausted.

3. Characters are decoded by routine P0A which also detects any illegal characters by the INPUT routine. R0 is assigned as the character buffer pointer and R4 is designated as the buffer size limit. The commands which affect the buffer size will affect R0 and R4.

For those readers who use the 8295 as a design example for UPI software, the flow charts for the program are shown in Figure 16 and the 8295 source listing is included in Appendix A. (Machine readable source listings are available through Intel's Intel User's Library.) As an aid to understanding the software, the following conventions can be made:

1. The 8295 uses only Register Bank 0. The function of registers R0 and R1 is determined by the mode. In parallel mode they are concatenated to form the 16 bit DMA count register. In serial mode, R0 is a counter during character reception.

APPENDIX A

PAGE 1

1212-11 REG-48V-1-40 WORD REGISTER, VS 8
LNC 1048 SERIES PRINTER CONTROLLER SOURCE CODE

LOC	OBJ	256	SPACE STATEMENT
1			1. WORDS TITLED LNC 1048 SERIES PRINTER CONTROLLER SOURCE CODE
2			2.
3			3.
4			4. 1048 CODE - LNC 1048 SERIES PRINTER CONTROLLER
5			5. 1048 CODE - LNC 1048 SERIES PRINTER CONTROLLER
6			6.
7			7.
8			8.
9			9.
10			10. COMMENT (C) 1048
11			11. INTEL COMMENT
12			12. 1048 CODE
13			13. 1048 CODE
14			14.
15			15.
16			16.
17			17.
18			18. 1048 CODE
19			19. 1048 CODE
20			20. 1048 CODE
21			21. 1048 CODE
22			22. 1048 CODE
23			23. 1048 CODE
24			24. 1048 CODE
25			25. 1048 CODE
26			26. 1048 CODE
27			27. 1048 CODE
28			28. 1048 CODE
29			29. 1048 CODE
30			30. 1048 CODE
31			31. 1048 CODE
32			32. 1048 CODE
33			33. 1048 CODE
34			34. 1048 CODE
35			35. 1048 CODE
36			36. 1048 CODE
37			37. 1048 CODE
38			38. 1048 CODE
39			39. 1048 CODE
40			40. 1048 CODE
41			41. 1048 CODE
42			42. 1048 CODE
43			43. 1048 CODE
44			44. 1048 CODE
45			45. 1048 CODE
46			46. 1048 CODE
47			47. 1048 CODE
48			48. 1048 CODE
49			49. 1048 CODE
50			50. 1048 CODE
51			51. 1048 CODE
52			52. 1048 CODE
53			53. 1048 CODE
54			54. 1048 CODE
55			55. 1048 CODE
56			56. 1048 CODE
57			57. 1048 CODE
58			58. 1048 CODE
59			59. 1048 CODE
60			60. 1048 CODE
61			61. 1048 CODE
62			62. 1048 CODE
63			63. 1048 CODE
64			64. 1048 CODE
65			65. 1048 CODE
66			66. 1048 CODE
67			67. 1048 CODE
68			68. 1048 CODE
69			69. 1048 CODE
70			70. 1048 CODE
71			71. 1048 CODE
72			72. 1048 CODE
73			73. 1048 CODE
74			74. 1048 CODE
75			75. 1048 CODE
76			76. 1048 CODE
77			77. 1048 CODE
78			78. 1048 CODE
79			79. 1048 CODE
80			80. 1048 CODE
81			81. 1048 CODE
82			82. 1048 CODE
83			83. 1048 CODE
84			84. 1048 CODE
85			85. 1048 CODE
86			86. 1048 CODE
87			87. 1048 CODE
88			88. 1048 CODE
89			89. 1048 CODE
90			90. 1048 CODE
91			91. 1048 CODE
92			92. 1048 CODE
93			93. 1048 CODE
94			94. 1048 CODE
95			95. 1048 CODE
96			96. 1048 CODE
97			97. 1048 CODE
98			98. 1048 CODE
99			99. 1048 CODE
100			100. 1048 CODE

PAGE 2

1212-11 REG-48V-1-40 WORD REGISTER, VS 8
LNC 1048 SERIES PRINTER CONTROLLER SOURCE CODE

LOC	OBJ	256	SPACE STATEMENT
101			101. 1048 CODE
102			102. 1048 CODE
103			103. 1048 CODE
104			104. 1048 CODE
105			105. 1048 CODE
106			106. 1048 CODE
107			107. 1048 CODE
108			108. 1048 CODE
109			109. 1048 CODE
110			110. 1048 CODE
111			111. 1048 CODE
112			112. 1048 CODE
113			113. 1048 CODE
114			114. 1048 CODE
115			115. 1048 CODE
116			116. 1048 CODE
117			117. 1048 CODE
118			118. 1048 CODE
119			119. 1048 CODE
120			120. 1048 CODE
121			121. 1048 CODE
122			122. 1048 CODE
123			123. 1048 CODE
124			124. 1048 CODE
125			125. 1048 CODE
126			126. 1048 CODE
127			127. 1048 CODE
128			128. 1048 CODE
129			129. 1048 CODE
130			130. 1048 CODE
131			131. 1048 CODE
132			132. 1048 CODE
133			133. 1048 CODE
134			134. 1048 CODE
135			135. 1048 CODE
136			136. 1048 CODE
137			137. 1048 CODE
138			138. 1048 CODE
139			139. 1048 CODE
140			140. 1048 CODE
141			141. 1048 CODE
142			142. 1048 CODE
143			143. 1048 CODE
144			144. 1048 CODE
145			145. 1048 CODE
146			146. 1048 CODE
147			147. 1048 CODE
148			148. 1048 CODE
149			149. 1048 CODE
150			150. 1048 CODE
151			151. 1048 CODE
152			152. 1048 CODE
153			153. 1048 CODE
154			154. 1048 CODE
155			155. 1048 CODE
156			156. 1048 CODE
157			157. 1048 CODE
158			158. 1048 CODE
159			159. 1048 CODE
160			160. 1048 CODE
161			161. 1048 CODE
162			162. 1048 CODE
163			163. 1048 CODE
164			164. 1048 CODE
165			165. 1048 CODE
166			166. 1048 CODE
167			167. 1048 CODE
168			168. 1048 CODE
169			169. 1048 CODE
170			170. 1048 CODE
171			171. 1048 CODE
172			172. 1048 CODE
173			173. 1048 CODE
174			174. 1048 CODE
175			175. 1048 CODE
176			176. 1048 CODE
177			177. 1048 CODE
178			178. 1048 CODE
179			179. 1048 CODE
180			180. 1048 CODE
181			181. 1048 CODE
182			182. 1048 CODE
183			183. 1048 CODE
184			184. 1048 CODE
185			185. 1048 CODE
186			186. 1048 CODE
187			187. 1048 CODE
188			188. 1048 CODE
189			189. 1048 CODE
190			190. 1048 CODE
191			191. 1048 CODE
192			192. 1048 CODE
193			193. 1048 CODE
194			194. 1048 CODE
195			195. 1048 CODE
196			196. 1048 CODE
197			197. 1048 CODE
198			198. 1048 CODE
199			199. 1048 CODE
200			200. 1048 CODE

APPENDIX A

ASM48 :F1:8295.SRC

ISIS-II MCS-48/UPI-41 MACRO ASSEMBLER, V2.0

PAGE 1

LRC 7040 SERIES PRINTER CONTROLLER SOURCE CODE

LOC	OBJ	SEQ	SOURCE STATEMENT
1			\$MOD42 TITLE('LRC 7040 SERIES PRINTER CONTROLLER SOURCE CODE')
2			
3			*****
4			** 8295 - LRC 7040 SERIES PRINTER CONTROLLER **
5			** REV. 0 FOR 7X7 CHARACTER MATRIX **
6			*****
7			
8			
9			
10			; COPYRIGHT (C) 1978
11			; INTEL CORPORATION
12			; 3065 BOWERS AVE.
13			; SANTA CLARA, CA. 95051
14			
15			
16			
17			*****
18			** PAGE0 CONTAINS THE INITIALIZATION SEQUENCE, THE OUTPUTING **
19			** OF DATA TO THE SOLENIOS, THE SERIAL INPUT ROUTINE, THE **
20			** PAPER FEED ROUTINE, AND THE SOLENIOD FIRETIME ROUTINE **
21			*****
22			
23			\$EJECT

ISIS-II MCS-48/UPI-41 MACRO ASSEMBLER, V2.0

PAGE 2

LRC 7040 SERIES PRINTER CONTROLLER SOURCE CODE

LOC	OBJ	SEQ	SOURCE STATEMENT
24			
25			
26			
27			*****
28			** **
29			; REGISTER ASSIGNMENT TABLE **
30			** **
31			*****
32			** **
33			R0 INPUT BUFFER POINTER **
34			R1 TEMPORARY STORAGE **
35			R2 TEMPORARY STORAGE **
36			R3 COMMAND REGISTER **
37			R4 BUFFER SIZE **
38			R5 TEMPORARY STORAGE FOR DELAY ROUTINE **
39			R6 LOW ORDER DMA COUNTER **
40			R7 HIGH ORDER DMA COUNTER **
41			TIMER TEMPORARY STORAGE **
42			** **
43			*****
44			
45			
46			\$EJECT

LOC	OBJ	SEQ	SOURCE STATEMENT
47			*****
48			**
49			**
50			RAM ASSIGNMENT TABLE
51			*****
52			**
53			RAM ADDRESS FUNCTION
54			**
55			00-07H REGISTER BANK 1
56			08-14H PROGRAM STACK
57			15-17H TAB POSITION STORAGE
58			18-40H CHARACTER BUFFER
59			**
60			*****
61			
62			\$EJECT

LOC	OBJ	SEQ	SOURCE STATEMENT
63			*****
64			**
65			**
66			COMMAND REGISTER DEFINITION
67			**
68			*****
69			**
70			BIT 7 SERIAL MODE FLAG
71			BIT 6 DMA MODE FLAG
72			BIT 5 DOUBLE WIDE FLAG
73			BIT 4 32 COLUMNS/LINE
74			BIT 3 RIGHT JUSTIFIED PRINT
75			BITS 2,1,0 INDICATE SOLENOID ON TIME
76			**
77			*****
78			\$EJECT

```

0000      79      ORG      000H
0000      80
0000      81
0000 02      82 INIT:   OUT      DBB,A ;SET OBF
0001 0A      83      IN      R,P2 ;CHECK SERIAL STRAP
0002 B208    84      JBS     PARA
0004 B803    85      MOV     R3,#03H ;SET SERIAL BIT IN CMD
0006 040E    86      JMP     CLR1
0008 9ABF    87 PARA:   ANL     P2,#0BFH
000A F5      88      EN      FLAGS
000B E5      89      EN      DMA
000C B803    90      MOV     R3,#03H
000E 27      91 CLR1:   CLR     A ;CLEAR DMA BUSY FLAG
000F 90      92      MOV     STS,A
0010 BC40    93 CLEAR:  MOV     R4,#40H ;INITIALIZE BUFFER
0012 B818    94 AGAIN:  MOV     R0,#18H ;INITIALIZE POINTER
0014 27      95      CLR     A ;RESET STACK TO SAVE TABS
0015 07      96      MOV     PSW,A ;STACK = 0, ALL FLAGS = 0
0016 3414    97 DECO    CALL    INPUT
0018 3428    98      CALL    PGR ;DECODE DATA
001A FC      99      MOV     R,R4
001B 08      100     XRL     A,R0
001C 9616    101     JNZ     DECO
001E FD      102
001F 08      103 PRINT:  MOV     A,R3
0020 7224    104     DEC     R0 ;LOCATE LAST CHARACTER INPUT IF R.J.
0022 B818    105     JBE     ON ;CHECK FOR RIGHT JUST.
0024 9AEF    106     MOV     R0,#18H ;PRINT FROM THE ORIGIN
0026 4626    107 ON:    ANL     P2,#0EFH ;TURN DRIVE MOTOR ON
0028 2340    108 NHOME:  JNT1    NHOME ;WAIT FOR HOME SWITCH
002A 54F8    109     MOV     A,#40H ;STALL
002C B006    110     CALL    WAIT
002E FB      111 XFER:   MOV     R2,#06H ;R.J. COL. INDEX
002F 7233    112     MOV     A,R3 ;CHECK FOR R.J.
0031 B000    113     JBE     CHAR ;R.J. TRUE
0033 F0      114     MOV     R2,#00H ;INDEX FOR NORM. PRINTING
0034 85      115 CHAR:  MOV     A,R0 ;FETCH CHARACTER
0035 B238    116     CLR     F0 ;F0 DETERMINES WHICH CHARACTER TABLE
0037 95      117     JBS     PAGE
0038 54E0    118     CPL     F0
003A A9      119 PAGE:  CALL    XS2 ;FETCH COL. FROM TABLE
003B FB      120     MOV     R1,A
003C B23F    121     MOV     A,R3 ;CHECK FOR D.W.
003E 95      122     JBS     NOTS
003F F9      123     CPL     F0 ;F0 INDICATES D.W. MODE
0040 147B    124 NOTS:  MOV     A,R1
0042 FB      125     CALL    FIRE ;PRINT COL.
0043 7240    126     MOV     A,R3 ;CHECK R.J.
0045 2306    127     JBE     RJP
0047 DA      128     MOV     A,#06H
0048 1A      129     XRL     A,R2
0049 9633    130     INC     R2
004B 0452    131     JNZ     CHAR ;PRINT NEXT COL.
004D 27      132     JMP     LSTCOL
004D 27      133 RJP:   CLR     A ;CHECK R.J. FINE COLS. IN REVERSE ORDER

```


LOC	OBJ	SEQ	SOURCE STATEMENT
004E	DA	134	XRL A,R2
004F	CA	135	DEC R2
0050	9633	136	JNZ CHAR
0052	B656	137	LSTCOL: JF0 A4
0054	1480	138	CALL COL8
0056	237F	139	A4: MOV A,#7FH ;CLEAR STB & DATA PINS
0058	39	140	OUTL P1,A
0059	2319	141	MOV A,#19H
005B	54F8	142	CALL WAIT
005D	FB	143	MOV A,R3
005E	7264	144	JB3 RJ2
0060	FC	145	MOV A,R4
0061	18	146	INC R0 ;INCR POINTER
0062	0467	147	JMP CK
0064	2317	148	RJ2: MOV A,#17H
0066	C8	149	DEC R0 ;DECR POINTER
0067	D8	150	CK: XRL A,R0
0068	962C	151	JNZ XFER ;RETURN FOR NEXT CHAR
006A	566A	152	HOME: JT1 HOME ;SENSE HOME LOW?
006C	2320	153	MOV A,#20H ;STALL
006E	54F8	154	CALL WAIT
0070	8A10	155	ORL P2,#10H ;STOP DRIVE MOTOR
0072	0412	156	JMP AGAIN ;NEXT LINE
		157	
0074	FB	158	DMAIN: MOV A,R3 ;EXIT IF SERIAL MODE
0075	F27A	159	JB7 SERR0R ;SERIAL CMD ERROR
0077	D677	160	INBUF: JNIBF INBUF ;WAIT FOR DMA PARAMS
0079	22	161	IN A,DBB
007A	93	162	SERR0R: RETR
		163	
		164	
		165	
007B	B67F	166	FIRE: JF0 SGLE
007D	09	167	IN A,P1 ;D.W. AND PREVIOUS COL.
007E	59	168	ANL A,R1
007F	39	169	SGLE: OUTL P1,A ;OUTPUT TO SOL.
0080	FB	170	COL8: MOV A,R3 ;A GETS ON TIME
0081	43F8	171	ORL A,#0F8H
0083	A3	172	MOVP A,0A
0084	530F	173	ANL A,#0FH
0086	8980	174	ORL P1,#80H ;STROBE SOLENOIDS
0088	54F8	175	CALL WAIT
008A	997F	176	ANL P1,#7FH ;DISABLE SOL. STROBE
008C	FB	177	MOV A,R3 ;A GET OFF TIME
008D	43F8	178	ORL A,#0F8H
008F	A3	179	MOVP A,0A
0090	47	180	SWAP A
0091	530F	181	ANL A,#0FH
0093	2B	182	XCH A,R3
0094	9299	183	JB4 C10
0096	2B	184	XCH A,R3
0097	049C	185	JMP CON
0099	2B	186	C10: XCH A,R3
009A	0306	187	ADD A,#06H ;INCREASE BIAS FOR 10C/I
009C	B6A3	188	CON: JF0 SING ;SKIP IF SINGLE

LOC	OBJ	SEQ	SOURCE STATEMENT
009E	0314	189	ADD A, #14H ; ADD 7 TO OFFTIME IF D.W.
00A0	29	190	XCH A, R1 ; SAVE PREVIOUS COL.
00A1	39	191	OUTL P1, A ; SAVE PREVIOUS COL.
00A2	29	192	XCH A, R1 ; RESTORE PREVIOUS COL.
00A3	44F8	193	SING JMP WAIT
		194	
		195	
		196	*****
		197	; SERIAL ROUTINE, ASSEMBLES THE DESIRED DATA FROM THE
		198	; SERIAL INPUT AND PLACE THE DATA IN THE ACCUMULATOR.
		199	*****
		200	
00A5	9ABF	201	CTS: ANL P2, #0BFH ; REQUEST /CTS
00A7	0A	202	ONE IN A, P2 ; LOOP UNTIL START BIT FOUND
00A8	F2A7	203	JB7 ONE ;
00AA	B900	204	MOV R1, #0 ; RESET TEMP REG
00AC	BA09	205	MOV R2, #09H ; SET INDEX
00AE	09	206	IN A, P1 ; BIAS
00AF	74E0	207	CALL HBIT ; WAIT 1/2 CYCLE
00B1	0A	208	IN A, P2 ; CHECK FOR START BIT
00B2	F2A7	209	JB7 ONE ; WRONG START BIT
00B4	BE03	210	MOV R6, #03H
00B6	EEB6	211	LZ: DJNZ R6, LZ
00B8	EACE	212	CONT: DJNZ R2, LOAD ; LOAD THE EIGHT BITS
00BA	8A40	213	ORL P2, #40H ; DISABLE /CTS
00BC	BE06	214	MOV R6, #06H ; BIAS
00BE	EEBE	215	W14: DJNZ R6, W14 ; WAIT
00C0	74E0	216	CALL HBIT
00C2	74E0	217	CALL HBIT
00C4	0A	218	IN A, P2
00C5	37	219	CPL A ; CHECK STOP BIT
00C6	F2A7	220	JB7 ONE ; WRONG STOP BIT
00C8	F9	221	MOV R1, A
00C9	F7	222	RLC A
00CA	537F	223	ANL A, #7FH
00CC	AA	224	MOV R2, A
00CD	93	225	RETR
		226	
		227	
00CE	74E0	228	LOAD: CALL HBIT ; DELAY 1 CYCLE
00D0	74E0	229	CALL HBIT
00D2	BE03	230	MOV R6, #03H
00D4	EED4	231	L1: DJNZ R6, L1
00D6	00	232	NOP
00D7	0A	233	IN A, P2 ; INPUT SERIAL BIT
00D8	5380	234	ANL A, #80H ; MASK BIT
00DA	49	235	ORL A, R1 ; ADD PREVIOUS BITS
00DB	67	236	RRC A
00DC	A9	237	MOV R1, A
00DD	04B8	238	JMP CONT ; FINISH JOB
		239	
00DF	9AFE	240	PF: ANL P2, #0FEH ; PF MOTOR ON
00E1	B90A	241	MOV R1, #0AH
00E3	2388	242	P3C: MOV A, #088H
00E5	54F8	243	CALL WAIT

LOC	OBJ	SEQ	SOURCE STATEMENT	DISPATCH	BRIDGE	482	180	001
00E7	E9E3	244	DJNZ R1,P3C					
00E9	F8	245	IT0: MOV A,R0 ;DELAY CONTANT =BUFF POINTER (18H TO 40H)					
00EA	26EA	246	IT1: JNT0 IT1					
00EC	54F8	247	CALL WAIT ;DELAY =1MS TO 2.5MS					
00EE	36E9	248	JT0 IT0					
00F0	23F3	249	MOV A,#0F3H ;STALL					
00F2	54F8	250	CALL WAIT					
00F4	8A01	251	ORL P2,#01H ;PF MOTOR OFF					
00F6	93	252	P3F: RETR					
		253						
00F8		254	ORG 0F8H ;SOL ON TIME CONSTANTS					
00F8	D4	255	DB 0D4H ;200US ON TIME					
00F9	C5	256	DB 0C5H ;240					
00FA	B6	257	DB 0B6H ;280					
00FB	A7	258	DB 0A7H ;320 ;DEFAULT					
00FC	98	259	DB 98H ;360					
00FD	89	260	DB 89H ;400					
00FE	7A	261	DB 7AH ;440					
00FF	6B	262	DB 6BH ;480					
		263						
		264						
		265	*****					
		266	: PAGE 1 INPUTS, DECODES, AND EXECUTES COMMANDS AND DATA					
		267	*****					
		268						
0100		269	ORG 100H					
0100	00	270	NOP					
0101	B5	271	DB (S01 AND 0FFH) ;ADDRESS FOR SET OUTPUT 1					
0102	B2	272	DB (S02 AND 0FFH) ;S02					
0103	BB	273	DB (R01 AND 0FFH) ;R01					
0104	B8	274	DB (R02 AND 0FFH) ;R02					
0105	BE	275	DB (RESET AND 0FFH) ;RESET					
0106	A8	276	DB (B32 AND 0FFH) ;B32					
0107	E4	277	DB (B40 AND 0FFH) ;B40					
0108	EA	278	DB (DWDE AND 0FFH) ;DWDE					
0109	C9	279	DB (SDMA AND 0FFH) ;SDMA					
010A	A0	280	DB (SSOL AND 0FFH) ;SSOL					
010B	88	281	DB (SLF AND 0FFH) ;SLF					
010C	81	282	DB (MLF AND 0FFH) ;MLF					
010D	84	283	DB (TOF AND 0FFH) ;TOF					
010E	DE	284	DB (CR AND 0FFH) ;CR					
010F	72	285	DB (T1 AND 0FFH) ;T1					
0110	72	286	DB (T2 AND 0FFH) ;T2					
0111	72	287	DB (T3 AND 0FFH) ;T3					
0112	F9	288	DB (RJ AND 0FFH) ;RJ					
0113	A0	289	DB (SSOL AND 0FFH) ;SSOL					
		290						
		291						
		292						
0114	FB	293	INPUT: MOV A,R3					
0115	F226	294	JB7 YME					
0117	37	295	CPL A					
0118	D21C	296	JB6 NODECR					
011A	8A40	297	ORL P2,#40H ;SET DRQ FOR DMA					
011C	D61C	298	NODECR: JNIBF NODECR ;SHARED BY PARALLEL & DMA					

LOC	OBJ	SEQ	SOURCE STATEMENT	DESTINATION	STATUS	LOC	OBJ
011E 22		299	IN A, DBB	011E 22	OK	011E 22	
011F 537F		300	ANL A, #7FH	011F 537F	OK	011F 537F	
0121 AA		301	MOV R2, A	0121 AA	OK	0121 AA	
0122 3462		302	CALL DECR ; DEC DMA COUNT FOR DMA & PARALLEL	0122 3462	OK	0122 3462	
0124 FA		303	MOV A, R2 ; DATA STORED IN A & R2	0124 FA	OK	0124 FA	
0125 93		304	RETR ; RET & RESTORE FLAGS	0125 93	OK	0125 93	
0126 04A5		305 YME:	JMP CTS ; SERIAL, USE SERIAL INPUT ROUTINE	0126 04A5	OK	0126 04A5	
		306					
0128 74ED		307 P6A:	CALL SPCR ; CHECK FOR SPECIAL CASE CR	0128 74ED	OK	0128 74ED	
012A D24E		308	JB6 CHECK5	012A D24E	OK	012A D24E	
012C B250		309	JB5 DATA ; CHECK FOR VALID CHAR	012C B250	OK	012C B250	
012E D309		310	XRL A, #09H ; TAB ?	012E D309	OK	012E D309	
0130 9656		311	JNZ CMD ; COMMAND	0130 9656	OK	0130 9656	
0132 B915		312 TAB:	MOV R1, #15H ; R1 GETS TAB[1]	0132 B915	OK	0132 B915	
0134 BA03		313	MOV R2, #03H	0134 BA03	OK	0134 BA03	
0136 F1		314 P6BB:	MOV A, @R1 ; CHECK TAB	0136 F1	OK	0136 F1	
0137 F240		315	JB7 TERROR ; LIMIT TAB TO DMAX	0137 F240	OK	0137 F240	
0139 D240		316	JB6 TERROR	0139 D240	OK	0139 D240	
013B 37		317	CPL A	013B 37	OK	013B 37	
013C 17		318	INC A	013C 17	OK	013C 17	
013D 68		319	ADD A, R0	013D 68	OK	013D 68	
013E F1		320	MOV A, @R1 ; A GET TAB LOC	013E F1	OK	013E F1	
013F E645		321	JNC P6AA ; FIND WHICH TAB	013F E645	OK	013F E645	
0141 19		322	INC R1	0141 19	OK	0141 19	
0142 EA36		323	DJNZ R2, P6BB	0142 EA36	OK	0142 EA36	
0144 FC		324 SPRL:	MOV A, R4 ; EXCEED ALL TAB, FILL IN BLANKS	0144 FC	OK	0144 FC	
0145 AA		325 P6AA:	MOV R2, A	0145 AA	OK	0145 AA	
0146 B020		326 RTAB:	MOV @R0, #20H	0146 B020	OK	0146 B020	
0148 18		327	INC R0	0148 18	OK	0148 18	
0149 FA		328	MOV A, R2	0149 FA	OK	0149 FA	
014A D8		329	XRL A, R0 ; FILL IN BLANKS	014A D8	OK	014A D8	
014B 9646		330	JNZ RTAB	014B 9646	OK	014B 9646	
014D 93		331 TERROR:	RETR	014D 93	OK	014D 93	
		332					
014E B255		333 CHECK5:	JB5 SEND	014E B255	OK	014E B255	
0150 FA		334 DATA:	MOV A, R2	0150 FA	OK	0150 FA	
0151 A0		335	MOV @R0, A	0151 A0	OK	0151 A0	
0152 18		336	INC R0	0152 18	OK	0152 18	
0153 54ED		337	CALL PEON ; SET SPECIAL FLAG FOR LAST DATA CHARACTER	0153 54ED	OK	0153 54ED	
0155 93		338 SEND:	KETR	0155 93	OK	0155 93	
		339					
0156 B914		340 CMD:	MOV R1, #14H ; R1 EQ INDEX	0156 B914	OK	0156 B914	
0158 FA		341 P7C:	MOV A, R2 ; A GETS CMD	0158 FA	OK	0158 FA	
0159 17		342	INC A	0159 17	OK	0159 17	
015A D9		343	XRL A, R1	015A D9	OK	015A D9	
015B C660		344	JZ FOUND ; MATCH ?	015B C660	OK	015B C660	
015D E958		345	DJNZ R1, P7C	015D E958	OK	015D E958	
015F 93		346	RETR	015F 93	OK	015F 93	
		347					
0160 F9		348 FOUND:	MOV A, R1	0160 F9	OK	0160 F9	
0161 B3		349	JMP @A ; JUMP INDIRECT TO CMD ROUTINE	0161 B3	OK	0161 B3	
		350					
0162 FE		351 DECR:	MOV A, R6	0162 FE	OK	0162 FE	
0163 9670		352	JNZ LARS ; DEC R6, R7 AS REG PAIR, RET ON 0	0163 9670	OK	0163 9670	
0165 4F		353	ORL A, R7	0165 4F	OK	0165 4F	

LOC	OBJ	SEQ	SOURCE STATEMENT
0166	966F	354	JNZ NRST
0169	2B	355	XCH A, R3
0169	53BF	356	ANL A, #0BFH
016B	2B	357	XCH A, R3
016C	90	358	MOV STS, A
016D	8A20	359	ORL P2, #20H ;ENABLE INTERRUPT PIN
016F	CF	360	NRST: DEC R7
0170	CE	361	LAPS: DEC R6
0171	93	362	RETR
		363	
		364	
		365	*****
		366	; COMMAND LOOK UP TABLE
		367	*****
		368	
		369	
		370	T1: ; A = ADDR OF CMD JUMP IN CMD TABLE
		371	T2: ; A = F, 1 OR 0
0172	17	372	T3: INC A ; A=0, 1, OR 2H
0173	5303	373	ANL A, #03H ; MASK SIGNIFICANT BITS
0175	0315	374	ADD A, #15H ; ACCUM = 15, 16, OR 17H -(RAM LOCATIONS FOR TABS)
0177	62	375	STAB: MOV T, A ; TEMP STORAGE FOR TAB
0178	3414	376	CALL INPUT
017A	0318	377	ADD A, #18H
017C	A9	378	MOV P1, A
017D	42	379	MOV A, T
017E	29	380	XCH A, R1
017F	A1	381	MOV @R1, A
0180	93	382	PETR
		383	
0181	85	384	MLF: CLR F0 ; MULTIPLE LINE FEED
0182	248A	385	JMP LF
		386	
0184	97	387	TOF: CLR C ; TOP OF FORM
0185	A7	388	CPL C
0186	248A	389	JMP LF
		390	
0188	85	391	SLF: CLR F0 ; SINGLE LINE FEED
0189	95	392	CPL F0
018A	F69C	393	LF: JC P12B ; LFUTOF
018C	B693	394	JF0 P12A ; SINGLE LF
018E	3414	395	CALL INPUT
0190	AA	396	MOV R2, A
0191	C69B	397	JZ P12C
0193	14DF	398	P12A: CALL PF
0195	F69C	399	JC P12B
0197	B69B	400	JF0 P12C
0199	EA93	401	DJNZ R2, P12A ; DECR # OF LINES
019B	93	402	P12C: RETR
019C	0A	403	P12B: IN A, P2
019D	3293	404	JB1 P12A
019F	93	405	RETR
		406	
01A0	3414	407	SSOL: CALL INPUT ; FETCH SOL. ON TIME
01A2	2B	408	XCH A, R3

LOC	OBJ	SEQ	SOURCE STATEMENT
01E8	0410	464	JMP CLEAR
		465	
		466	
		467	
01EA	2320	468	DWDE: MOV A, #20H ; DOUBLE WIDE PRINT MODE
01EC	4B	469	ORL A, R3 ; SET DW BIT
01ED	AB	470	MOV R3, A
01EE	B818	471	MOV R0, #18H ; CLEAR BUFFER POINTER
01F0	FC	472	MOV A, R4
01F1	D2F6	473	JB6 X0
01F3	BC2A	474	MOV R4, #2AH ; 32 CHAR. BUFFER
01F5	93	475	RETR
01F6	BC2C	476	X0: MOV R4, #2CH ; 40 CHAR. BUFFER
01F8	93	477	RETR
		478	
01F9	FB	479	RJ: MOV A, R3 ; SET RJ BIT IN CMD
01FA	4308	480	ORL A, #08H
01FC	AB	481	MOV R3, A
01FD	93	482	RETR
		483	
		484	
		485	
		486	*****
		487	; HBIT SUBR. AND THE DATA CONSTANTS ARE IN PAGE 3
		488	*****
		489	
03E0		490	ORG 2E0H
		491	
03E0	22	492	HBIT: IN A, DBB ; CHECK DBB FOR BUAD RATE
03E1	43F8	493	ORL A, #0F8H
03E3	A3	494	MOVP A, @A
03E4	AE	495	MOV R6, A
03E5	BF03	496	LOOP1: MOV R7, #03H ; 25US PER LOOP PAIR
03E7	EFE7	497	LOOP2: DJNZ R7, LOOP2
03E9	EEE5	498	DJNZ R6, LOOP1
03EB	0A	499	IN A, P2
03EC	93	500	RETR
		501	
03ED	D30D	502	SPCR: XRL A, #0DH ; CHECK CR FLAG, EXIT IF TRUE
03EF	96F5	503	JNZ XCR
03F1	34DE	504	CALL CR
03F3	BAFF	505	MOV R2, #0FFH ; DO NOT EXECUTE CR TWICE
03F5	FA	506	XCR: MOV A, R2
03F6	62	507	MOV T, A
03F7	93	508	RETR
		509	
03F8		510	ORG 3F8H
		511	
03F8	B2	512	DB 0B2H ; 110 BAUD
03F9	84	513	DB 0B4H ; 150
03FA	40	514	DB 40H ; 300
03FB	1F	515	DB 1FH ; 600
03FC	0E	516	DB 0EH ; 1200
03FD	06	517	DB 06H ; 2400
03FE	02	518	DB 02H ; 4800

LOC	OBJ	SEQ	SOURCE STATEMENT
03FF	02	519	DB 02H ; 4800
		520	
		521	*****
		522	OTHER THAN CHAR TABLE, WAIT AND X52 ROUTINES EXIST IN PAGE 2
		523	*****
		524	
02E0		525	ORG 2E0H
02E0 531F		526	ANL A, #1FH ; FIND & ADJUST CHARACTER INDEX
02E2 A9		527	MOV R1, A ; MULTIPLY INDEX BY 7
02E3 E7		528	RL A
02E4 E7		529	RL A
02E5 69		530	ADD A, R1
02E6 69		531	ADD A, R1
02E7 69		532	ADD A, R1
02E8 6A		533	ADD A, R2 ; ADD COLUMN INDEX TO CHARACTER INDEX
02E9 B6F5		534	JF0 PAGE3
02EB E3		535	MOVP3 A, 0A
02EC 83		536	RET
02ED FC		537	PEON: MOV A, R4 ; SET SPECIAL OR FLAG IF LAST CHAR IS DATA
02EE D8		538	XRL A, R0
02EF 96F4		539	JNZ FSPA
02F1 230D		540	MOV A, #0DH
02F3 62		541	MOV T, A
02F4 93		542	FSPA: RET
		543	
		544	
02F5 A3		545	PAGE3: MOVP A, 0A
02F6 85		546	CLR F0
02F7 83		547	RET
02F8 BD06		548	WAIT: MOV R5, #06H
02FA EDFA		549	CONX: DJNZ R5, CONX ; 40US PER COUNT OF ACC
02FC 07		550	DEC A
02FD 96F8		551	JNZ WAIT
02FF 93		552	RETR
		553	
		554	
		555	
		556	*****
		557	CHARACTER TABLE IN PAGE 2.
		558	MSB IS IGNORED, DATA INVERTED
		559	SEE EXAMPLE (A)
		560	*****
		561	
0200		562	ORG 200H
		563	
0200 41		564	DB 41H ; 0
0201 3F		565	DB 3FH
0202 62		566	DB 62H
0203 3F		567	DB 3FH
0204 62		568	DB 62H
0205 3F		569	DB 3FH
0206 43		570	DB 43H
		571	
0207 70		572	DB 70H ; A ----*
0208 6F		573	DB 6FH ; 0 ----*

LOC	OBJ	SEQ	SOURCE STATEMENT
0209	5B	574	DB 5BH ; -----*
020A	3F	575	DB 3FH ; *-----
020B	5B	576	DB 5BH ; -----*
020C	6F	577	DB 6FH ; ---*-----
020D	70	578	DB 70H ; -----****
		579	
020E	3E	580	DB 3EH ;B
020F	41	581	DB 41H
0210	3E	582	DB 3EH
0211	77	583	DB 77H
0212	3E	584	DB 3EH
0213	77	585	DB 77H
0214	49	586	DB 49H
		587	
0215	41	588	DB 41H ;C
0216	3E	589	DB 3EH
0217	7F	590	DB 7FH
0218	3E	591	DB 3EH
0219	7F	592	DB 7FH
021A	3E	593	DB 3EH
021B	50	594	DB 50H
		595	
021C	3E	596	DB 3EH ;D
021D	41	597	DB 41H
021E	3E	598	DB 3EH
021F	7F	599	DB 7FH
0220	3E	600	DB 3EH
0221	7F	601	DB 7FH
0222	41	602	DB 41H
		603	
0223	00	604	DB 00H ;E
0224	7F	605	DB 7FH
0225	36	606	DB 36H
0226	7F	607	DB 7FH
0227	36	608	DB 36H
0228	7F	609	DB 7FH
0229	3E	610	DB 3EH
		611	
022A	00	612	DB 00H ;F
022B	7F	613	DB 7FH
022C	37	614	DB 37H
022D	7F	615	DB 7FH
022E	37	616	DB 37H
022F	7F	617	DB 7FH
0230	3F	618	DB 3FH
		619	
0231	41	620	DB 41H ;G
0232	3E	621	DB 3EH
0233	7F	622	DB 7FH
0234	3E	623	DB 3EH
0235	7B	624	DB 7BH
0236	3E	625	DB 3EH
0237	59	626	DB 59H
		627	
0238	00	628	DB 00H

LOC	OBJ	SEQ	SOURCE STATEMENT
0239	7F	629	DB 7FH ;H
023A	77	630	DB 77H
023B	7F	631	DB 7FH
023C	77	632	DB 77H
023D	7F	633	DB 7FH
023E	00	634	DB 00H
		635	
023F	7F	636	DB 7FH ;I
0240	3E	637	DB 3EH
0241	7F	638	DB 7FH
0242	00	639	DB 00H
0243	7F	640	DB 7FH
0244	3E	641	DB 3EH
0245	7F	642	DB 7FH
		643	
0246	7D	644	DB 7DH ;J
0247	7E	645	DB 7EH
0248	7F	646	DB 7FH
0249	7E	647	DB 7EH
024A	7F	648	DB 7FH
024B	7E	649	DB 7EH
024C	01	650	DB 01H
		651	
024D	00	652	DB 00H ;K
024E	7F	653	DB 7FH
024F	6F	654	DB 6FH
0250	77	655	DB 77H
0251	5B	656	DB 5BH
0252	7D	657	DB 7DH
0253	3E	658	DB 3EH
		659	
0254	00	660	DB 00H
0255	7F	661	DB 7FH
0256	7E	662	DB 7EH ;L
0257	7F	663	DB 7FH
0258	7E	664	DB 7EH
0259	7F	665	DB 7FH
025A	7E	666	DB 7EH
		667	
025B	40	668	DB 40H ;M
025C	3F	669	DB 3FH
025D	5F	670	DB 5FH
025E	67	671	DB 67H
025F	5F	672	DB 5FH
0260	3F	673	DB 3FH
0261	40	674	DB 40H
		675	
0262	20	676	DB 20H
0263	5F	677	DB 5FH ;N
0264	6F	678	DB 6FH
0265	77	679	DB 77H
0266	7B	680	DB 7BH
0267	7D	681	DB 7DH
0268	02	682	DB 02H
		683	

LOC	OBJ	SEQ	SOURCE STATEMENT
0269	41	684	DB 41H
026A	3E	685	DB 3EH
026B	7F	686	DB 7FH
026C	3E	687	DB 3EH
026D	7F	688	DB 7FH
026E	3E	689	DB 3EH
026F	41	690	DB 41H
		691	
0270	00	692	DB 00H
0271	7F	693	DB 7FH
0272	37	694	DB 37H
0273	7F	695	DB 7FH
0274	37	696	DB 37H
0275	7F	697	DB 7FH
0276	4F	698	DB 4FH
		699	
0277	41	700	DB 41H
0278	3E	701	DB 3EH
0279	7F	702	DB 7FH
027A	3F	703	DB 3FH
027B	7A	704	DB 7AH
027C	3D	705	DB 3DH
027D	42	706	DB 42H
		707	
027E	00	708	DB 00H
027F	7F	709	DB 7FH
0280	37	710	DB 37H
0281	7F	711	DB 7FH
0282	33	712	DB 33H
0283	7D	713	DB 7DH
0284	4E	714	DB 4EH
		715	
0285	4D	716	DB 4DH
0286	36	717	DB 36H
0287	7F	718	DB 7FH
0288	36	719	DB 36H
0289	7F	720	DB 7FH
028A	36	721	DB 36H
028B	59	722	DB 59H
		723	
028C	3F	724	DB 3FH
028D	7F	725	DB 7FH
028E	3F	726	DB 3FH
028F	40	727	DB 40H
0290	3F	728	DB 3FH
0291	7F	729	DB 7FH
0292	3F	730	DB 3FH
		731	
0293	01	732	DB 01H
0294	7E	733	DB 7EH
0295	7F	734	DB 7FH
0296	7E	735	DB 7EH
0297	7F	736	DB 7FH
0298	7E	737	DB 7EH
0299	01	738	DB 01H

LOC	OBJ	SEQ	SOURCE STATEMENT
		739	
029A	07	740	DB 07H
029B	7B	741	DB 7BH
029C	7D	742	DB 7DH
029D	7E	743	DB 7EH
029E	7D	744	DB 7DH
029F	7B	745	DB 7BH
02A0	07	746	DB 07H
		747	
02A1	01	748	DB 01H
02A2	7E	749	DB 7EH
02A3	7D	750	DB 7DH
02A4	73	751	DB 73H
02A5	7D	752	DB 7DH
02A6	7E	753	DB 7EH
02A7	01	754	DB 01H
		755	
02A8	3E	756	DB 3EH
02A9	5D	757	DB 5DH
02AA	6B	758	DB 6BH
02AB	77	759	DB 77H
02AC	6B	760	DB 6BH
02AD	5D	761	DB 5DH
02AE	3E	762	DB 3EH
		763	
02AF	3F	764	DB 3FH
02B0	5F	765	DB 5FH
02B1	6F	766	DB 6FH
02B2	70	767	DB 70H
02B3	6F	768	DB 6FH
02B4	5F	769	DB 5FH
02B5	3F	770	DB 3FH
		771	
02B6	3E	772	DB 3EH
02B7	7D	773	DB 7DH
02B8	3A	774	DB 3AH
02B9	77	775	DB 77H
02BA	2E	776	DB 2EH
02BB	5F	777	DB 5FH
02BC	3E	778	DB 3EH
		779	
02BD	00	780	DB 00H
02BE	7F	781	DB 7FH
02BF	3E	782	DB 3EH
02C0	7F	783	DB 7FH
02C1	3E	784	DB 3EH
02C2	7F	785	DB 7FH
02C3	7F	786	DB 7FH
		787	
02C4	3F	788	DB 3FH
02C5	5F	789	DB 5FH
02C6	6F	790	DB 6FH
02C7	77	791	DB 77H
02C8	7B	792	DB 7BH
02C9	7D	793	DB 7DH

LOC	OBJ	SEQ	SOURCE STATEMENT
02CA	7E	794	DB 02H 7EH
		795	
02CB	7F	796	DB 03H 7FH ; J
02CC	7F	797	DB 04H 7FH
02CD	3E	798	DB 05H 3EH
02CE	7F	799	DB 06H 7FH
02CF	3E	800	DB 07H 3EH
02D0	7F	801	DB 08H 7FH
02D1	00	802	DB 09H 00H
		803	
02D2	77	804	DB 0AH 77H ; 0
02D3	6F	805	DB 0BH 6FH
02D4	5F	806	DB 0CH 5FH
02D5	20	807	DB 0DH 20H
02D6	5F	808	DB 0EH 5FH
02D7	6F	809	DB 0FH 6FH
02D8	77	810	DB 10H 77H
		811	
02D9	7E	812	DB 11H 7EH ; 1
02DA	7F	813	DB 12H 7FH
02DB	7E	814	DB 13H 7EH
02DC	7F	815	DB 14H 7FH
02DD	7C	816	DB 15H 7EH
02DE	7F	817	DB 16H 7FH
02DF	7E	818	DB 17H 7EH
		819	
		820	
		821	
		822	*****
		823	CHAR. TABLE ON PAGE 3.
		824	MSB IS IGNORED, DATA INVERTED
		825	SEE EXAMPLE (A) IN PAGE 2 OF ROM
		826	*****
		827	
0300		828	ORG 300H
		829	
0300	7F	830	DB 00H 7FH ; BLANK
0301	7F	831	DB 01H 7FH
0302	7F	832	DB 02H 7FH
0303	7F	833	DB 03H 7FH
0304	7F	834	DB 04H 7FH
0305	7F	835	DB 05H 7FH
0306	7F	836	DB 06H 7FH
		837	
0307	7F	838	DB 07H 7FH ; !
0308	7F	839	DB 08H 7FH
0309	7F	840	DB 09H 7FH
030A	02	841	DB 0AH 02H
030B	7F	842	DB 0BH 7FH
030C	7F	843	DB 0CH 7FH
030D	7F	844	DB 0DH 7FH
		845	
030E	7F	846	DB 0EH 7FH ; "
030F	7F	847	DB 0FH 7FH
0310	0F	848	DB 10H 0FH

LOC	OBJ	SEQ	SOURCE STATEMENT
0311	7F	849	DB 11H 7FH
0312	0F	850	DB 12H 0FH
0313	7F	851	DB 13H 7FH
0314	7F	852	DB 14H 7FH
		853	
0315	6B	854	DB 15H 6BH ; #
0316	7F	855	DB 16H 7FH
0317	00	856	DB 17H 00H
0318	7F	857	DB 18H 7FH
0319	00	858	DB 19H 00H
031A	7F	859	DB 1AH 7FH
031B	6B	860	DB 1BH 6BH
		861	
031C	40	862	DB 1CH 40H ; \$
031D	36	863	DB 1DH 36H
031E	7F	864	DB 1EH 7FH
031F	00	865	DB 1FH 00H
0320	7F	866	DB 20H 7FH
0321	36	867	DB 21H 36H
0322	59	868	DB 22H 59H
		869	
0323	0E	870	DB 23H 0EH ; %
0324	7D	871	DB 24H 7DH
0325	0B	872	DB 25H 0BH
0326	77	873	DB 26H 77H
0327	6B	874	DB 27H 6BH
0328	5F	875	DB 28H 5FH
0329	38	876	DB 29H 38H
		877	
032A	49	878	DB 2AH 49H ; &
032B	36	879	DB 2BH 36H
032C	7F	880	DB 2CH 7FH
032D	37	881	DB 2DH 37H
032E	5A	882	DB 2EH 5AH
032F	7D	883	DB 2FH 7DH
0330	72	884	DB 30H 72H
		885	
0331	7F	886	DB 31H 7FH ; '
0332	7F	887	DB 32H 7FH
0333	7F	888	DB 33H 7FH
0334	0F	889	DB 34H 0FH
0335	7F	890	DB 35H 7FH
0336	7F	891	DB 36H 7FH
0337	7F	892	DB 37H 7FH
		893	
0338	7F	894	DB 38H 7FH
0339	63	895	DB 39H 63H ; (
033A	5D	896	DB 3AH 5DH
033B	3E	897	DB 3BH 3EH
033C	7F	898	DB 3CH 7FH
033D	7F	899	DB 3DH 7FH
033E	7F	900	DB 3EH 7FH
		901	
033F	7F	902	DB 3FH 7FH ;)
0340	7F	903	DB 40H 7FH

LOC	OBJ	SEQ	SOURCE STATEMENT
0341	7F	904	DB 7FH ;1
0342	3E	905	DB 3EH ;2
0343	5D	906	DB 5DH ;3
0344	63	907	DB 63H ;4
0345	7F	908	DB 7FH ;5
		909	
0346	77	910	DB 77H ;6
0347	5D	911	DB 5DH ;7
0348	6B	912	DB 6BH ;8
0349	14	913	DB 14H ;9
034A	6B	914	DB 6BH ;10
034B	5D	915	DB 5DH ;11
034C	77	916	DB 77H ;12
		917	
034D	77	918	DB 77H ;13
034E	7F	919	DB 7FH ;14
034F	77	920	DB 77H ;15
0350	49	921	DB 49H ;16
0351	77	922	DB 77H ;17
0352	7F	923	DB 7FH ;18
0353	77	924	DB 77H ;19
		925	
0354	7F	926	DB 7FH ;20
0355	7F	927	DB 7FH ;21
0356	7F	928	DB 7FH ;22
0357	7E	929	DB 7EH ;23
0358	79	930	DB 79H ;24
0359	7F	931	DB 7FH ;25
035A	7F	932	DB 7FH ;26
		933	
035B	7B	934	DB 7BH ;27
035C	7F	935	DB 7FH ;28
035D	7B	936	DB 7BH ;29
035E	7F	937	DB 7FH ;30
035F	7B	938	DB 7BH ;31
0360	7F	939	DB 7FH ;32
0361	7B	940	DB 7BH ;33
		941	
0362	7F	942	DB 7FH ;34
0363	7F	943	DB 7FH ;35
0364	7F	944	DB 7FH ;36
0365	7E	945	DB 7EH ;37
0366	7F	946	DB 7FH ;38
0367	7F	947	DB 7FH ;39
0368	7F	948	DB 7FH ;40
		949	
0369	7E	950	DB 7EH ;41
036A	7D	951	DB 7DH ;42
036B	7B	952	DB 7BH ;43
036C	77	953	DB 77H ;44
036D	6F	954	DB 6FH ;45
036E	5F	955	DB 5FH ;46
036F	3F	956	DB 3FH ;47
		957	
0370	41	958	DB 41H ;48

LOC	OBJ	SEQ	SOURCE STATEMENT
0371	7F	959	DB 7FH ;49
0372	3A	960	DB 3AH ;50
0373	77	961	DB 77H ;51
0374	2E	962	DB 2EH ;52
0375	7F	963	DB 7FH ;53
0376	41	964	DB 41H ;54
		965	
0377	7F	966	DB 7FH ;55
0378	5E	967	DB 5EH ;56
0379	7F	968	DB 7FH ;57
037A	00	969	DB 00H ;58
037B	7F	970	DB 7FH ;59
037C	7E	971	DB 7EH ;60
037D	7F	972	DB 7FH ;61
		973	
037E	5C	974	DB 5CH ;62
037F	3B	975	DB 3BH ;63
0380	7E	976	DB 7EH ;64
0381	37	977	DB 37H ;65
0382	7E	978	DB 7EH ;66
0383	37	979	DB 37H ;67
0384	4E	980	DB 4EH ;68
		981	
0385	3D	982	DB 3DH ;69
0386	7E	983	DB 7EH ;70
0387	7F	984	DB 7FH ;71
0388	7E	985	DB 7EH ;72
0389	2F	986	DB 2FH ;73
038A	56	987	DB 56H ;74
038B	39	988	DB 39H ;75
		989	
038C	7B	990	DB 7BH ;76
038D	77	991	DB 77H ;77
038E	6B	992	DB 6BH ;78
038F	5F	993	DB 5FH ;79
0390	20	994	DB 20H ;80
0391	7F	995	DB 7FH ;81
0392	7B	996	DB 7BH ;82
		997	
0393	00	998	DB 00H ;83
0394	7E	999	DB 7EH ;84
0395	2F	1000	DB 2FH ;85
0396	7E	1001	DB 7EH ;86
0397	3F	1002	DB 3FH ;87
0398	6E	1003	DB 6EH ;88
0399	31	1004	DB 31H ;89
		1005	
039A	79	1006	DB 79H ;90
039B	76	1007	DB 76H ;91
039C	6F	1008	DB 6FH ;92
039D	56	1009	DB 56H ;93
039E	3F	1010	DB 3FH ;94
039F	76	1011	DB 76H ;95
03A0	79	1012	DB 79H ;96
		1013	

LOC	OBJ	SEQ	SOURCE STATEMENT
03A1	3F	1014	DB 3FH ;7
03A2	7F	1015	DB 7FH
03A3	38	1016	DB 38H
03A4	77	1017	DB 77H
03A5	2F	1018	DB 2FH
03A6	5F	1019	DB 5FH
03A7	3F	1020	DB 3FH
		1021	
03A8	49	1022	DB 49H ;8
03A9	36	1023	DB 36H
03AA	7F	1024	DB 7FH
03AB	36	1025	DB 36H
03AC	7F	1026	DB 7FH
03AD	36	1027	DB 36H
03AE	49	1028	DB 49H
		1029	
03AF	4F	1030	DB 4FH ;9
03B0	37	1031	DB 37H
03B1	7F	1032	DB 7FH
03B2	36	1033	DB 36H
03B3	7D	1034	DB 7DH
03B4	38	1035	DB 38H
03B5	47	1036	DB 47H
		1037	
03B6	7F	1038	DB 7FH
03B7	7F	1039	DB 7FH ;:
03B8	7F	1040	DB 7FH
03B9	6D	1041	DB 6DH
03BA	7F	1042	DB 7FH
03BB	7F	1043	DB 7FH
03BC	7F	1044	DB 7FH
		1045	
03BD	7F	1046	DB 7FH ;;
03BE	7F	1047	DB 7FH
03BF	7E	1048	DB 7EH
03C0	69	1049	DB 69H
03C1	7F	1050	DB 7FH
03C2	7F	1051	DB 7FH
03C3	7F	1052	DB 7FH
		1053	
03C4	7F	1054	DB 7FH ;<
03C5	77	1055	DB 77H
03C6	68	1056	DB 68H
03C7	5D	1057	DB 5DH
03C8	3E	1058	DB 3EH
03C9	7F	1059	DB 7FH
03CA	7F	1060	DB 7FH
		1061	
03CB	68	1062	DB 68H ;=
03CC	7F	1063	DB 7FH
03CD	68	1064	DB 68H
03CE	7F	1065	DB 7FH
03CF	68	1066	DB 68H
03D0	7F	1067	DB 7FH
03D1	68	1068	DB 68H

LOC	OBJ	SEQ	SOURCE STATEMENT
		1069	
03D2	7F	1070	DB 7FH ;>
03D3	7F	1071	DB 7FH
03D4	3E	1072	DB 3EH
03D5	5D	1073	DB 5DH
03D6	68	1074	DB 68H
03D7	77	1075	DB 77H
03D8	7F	1076	DB 7FH
		1077	
03D9	7F	1078	DB 7FH ;?
03DA	5F	1079	DB 5FH
03DB	3F	1080	DB 3FH
03DC	7A	1081	DB 7AH
03DD	37	1082	DB 37H
03DE	4F	1083	DB 4FH
03DF	7F	1084	DB 7FH
		1085	
		1086	END

ASSEMBLY COMPLETE, NO ERRORS

12-3	Pinout Evolution
12-4	System Architecture
12-5	Bus Contention
12-6	The Microprocessor/Memory Interface
12-7	Terminology
12-8	The New Intel Family
12-9	Pin Devices and Pin Sizes
12-10	Pinout Circuit Board Design

Application of Intel's 5V EPROM and ROM Family for Microprocessor Systems

Bob Greene
Application Engineering

Application of Intel's 5V EPROM and ROM Family for Microprocessor Systems

Contents

Introduction	12-3
Pinout Evolution	12-3
System Architecture	12-3
Bus Contention	12-4
The Microprocessor/Memory Interface	12-5
Terminology	12-5
The New Intel Family	12-6
Pin Devices and Pin Sites	12-7
Printed Circuit Board Design	12-8

INTRODUCTION

This Application Note discusses how the new Intel family of 5 volt EPROMs and ROMs can be used with microprocessor systems. The pinout evolution and philosophy are explored in detail, which leads directly to system architecture. Particular emphasis will be placed on the pitfalls of bus contention and the microprocessor/memory interface. Finally, an actual printed circuit board layout is presented.

PINOUT EVOLUTION

As EPROM/ROM technology has evolved, there are often periods of confusion over EPROM and ROM pinouts, as ROM density usually leads EPROM density by a factor of two, but ultimately users want any given EPROM to have a ROM compatible part. As we have seen, after the 2716 16K EPROM was introduced, a new ROM pinout emerged and "triumphed" over an earlier "standard." The reason this ROM pinout change occurred is that as codes stabilize in user's systems and equipment, many users opt for the less expensive ROMs, which are mask programmable devices. At the same time, users often use the highest available density ROM so they combine modular firmware and minimize device count. Of course, many users never do go to the ROM stage with their equipment, preferring to minimize inventory levels and utilize standard designs that can be customized for final equipment configurations, but they always want the capability to do so if desired.

In addition, over the past few years, the development of microprocessors has been intimately entwined with both ROMs and EPROMs.

The 1702A and its ROM counterpart, the 1302, were completely adequate to support the requirements of the 4004 series of microprocessors. In order to support the 5 volt, 3MHz 8085A and 5MHz 8086, it is desirable to use a compatible device such as the Intel 5 volt 2716, whose 450ns access time is compatible with the microprocessor requirements. Some high performance versions of these processors may require selected versions of the 2716 (such as the 2716-1 with $t_{ACC}=350ns$, or the 2716-2 with $t_{ACC}=390ns$) depending on the actual system configuration.

Summarizing these events since the introduction of the Intel 1702A, which was the first EPROM, we can postulate the following hypothesis: at any point in time, the present EPROM determines the pinout for the next generation ROM. And, if the subsequent larger density EPROM is not ROM compatible, the ROM will change. Also, it can be seen that ROMs and EPROMs must evolve along with microprocessor developments—so memory performance does not limit system performance.

The devices which are discussed in this Application Note represent an extension of the 5 volt compatible family to 32K bit and 64K bit densities, while improving performance as discussed above. It also follows that the pinout

for the 32K devices must be derived from the 2716 in order to maintain socket compatibility. This 16K to 32K pinout evolution is shown in Figure 1.

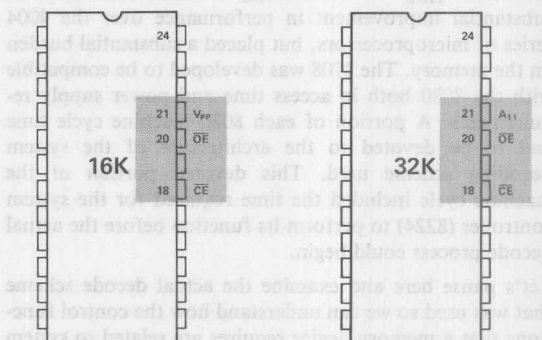


Figure 1. 16K EPROM Determines 32K ROM Pinout

SYSTEM ARCHITECTURE

As higher performance microprocessors have become available, the architecture of microprocessor systems has been evolving, again placing demands on memory. For many years, system designers have been plagued with the problem of bus contention when connecting multiple memories to a common data bus. There have been various schemes for avoiding the problem, but device manufacturers have been unable to design internal circuits that would guarantee that one memory device would be "off" the bus before another device was selected. With small memories (512x8 and 1Kx8), it has been traditional to connect all the system address lines together and utilize the difference between t_{ACC} and t_{CO} to perform a decode to select the correct device (as shown in Figure 2).

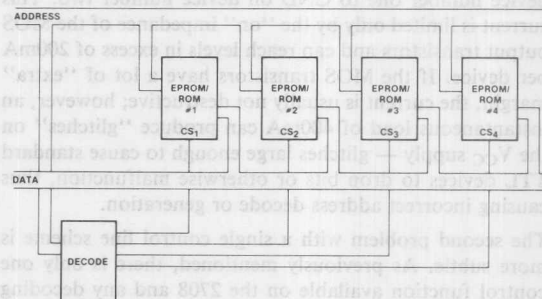


Figure 2. Single Control Line Architecture

processors for the 4004 series microprocessors, but the 8080 processor required that the corresponding numbers be reduced to $t_{ACC} = 450\text{ns}$ and $t_{CO} = 120\text{ns}$. This allowed a substantial improvement in performance over the 4004 series of microprocessors, but placed a substantial burden on the memory. The 2708 was developed to be compatible with the 8080 both in access time and power supply requirements. A portion of each 8080 machine cycle time had to be devoted to the architecture of the system decoding scheme used. This devoted portion of the machine cycle included the time required for the system controller (8224) to perform its function before the actual decode process could begin.

Let's pause here and examine the actual decode scheme that was used so we can understand how the control functions that a memory device requires are related to system architecture.

The 2708 can be used to illustrate the problem of having a single control line. The 2708 has only one read control function, chip select (\overline{CS}), which is very fast ($t_{CO} = 120\text{ns}$) with respect to the overall access time ($t_{ACC} = 450\text{ns}$) of the 2708. It is this time difference (330ns) that is used to perform the decode function, as illustrated in Figure 3. The scheme works well and does not limit system performance, but it does lead to the possibility of bus contention.

BUS CONTENTION

There are actually two problems with the scheme described in the previous section. First, if one device in a multiple memory system has a relatively long deselect time, and a relatively fast decoder is used, it would be possible to have another device selected at the same time. If the two devices thus selected were reading opposite data; that is, device number one reading a HIGH and device number two reading a LOW, the output transistors of the two memory devices would effectively produce a short circuit, as Figure 4 illustrates. In this case, the current path is from V_{CC} on device number one to GND on device number two. This current is limited only by the "on" impedance of the MOS output transistors and can reach levels in excess of 200mA per device. If the MOS transistors have a lot of "extra" margin, the current is usually not destructive; however, an instantaneous load of 400mA can produce "glitches" on the V_{CC} supply — glitches large enough to cause standard TTL devices to drop bits or otherwise malfunction, thus causing incorrect address decode or generation.

The second problem with a single control line scheme is more subtle. As previously mentioned, there is only one control function available on the 2708 and any decoding scheme must use it out of necessity. In addition, any inadvertent changes in the state of the high order address lines that are inputs to the decoder will cause a change in

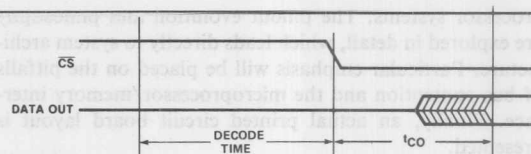


Figure 3. Single Line Control Architecture

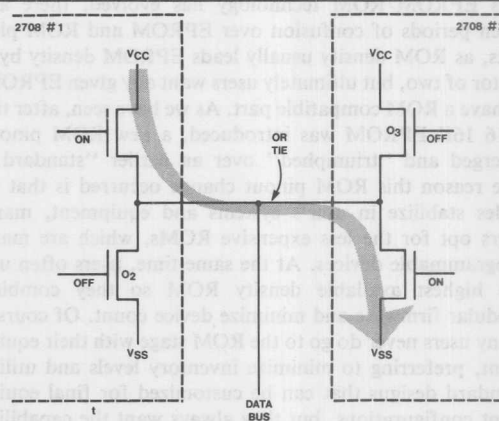


Figure 4. Results of Improper Timing when OR Tying Multiple Memories

the device that is selected. The result is the same as before — bus contention, only from a different source. The deselected device cannot get "off" the bus before the selected one is "on" the bus as the addresses rapidly change state. One approach to solving this problem would be to design (and specify as a maximum) devices with t_{DF} time less than t_{CO} time, thereby assuring that if one device is selected while another is simultaneously being deselected, there would be some small (20ns) margin. Even with this solution, the user would not be protected from devices which have very fast t_{CO} times (t_{CO} is specified as a maximum).

The only sure solution appears to be the use of an external bus driver/transceiver that has an independent enable function. Then that function, not the "device selecting function," or addresses, could control the flow of data "on" and "off" the bus, and any contention problems would be confined to a particular card or area of a large card. In fact, many systems are implemented that way — the use of bus drivers is not at all uncommon in large systems where the drive requirements of long, highly capacitive interconnecting lines must be taken into consideration — it also may be the reason why more system designers were not aware of the bus contention problem

until they took a previously large (multicard) system and, using an advanced microprocessor and higher density memory devices, combined them all on one card, thereby eliminating the requirement for the bus drivers, but experiencing the problem of bus contention as described above.

THE MICROPROCESSOR/MEMORY INTERFACE

From the foregoing discussion, it becomes clear that some new concepts, both with regard to architecture and performance are required. A new generation of two control line EPROM devices is called for with general requirements as listed below:

1. Complete ROM pin and function compatibility.
2. A power control function that allows the device to enter a low-power standby mode when deselected. This function can be used as the primary device selecting function, independent of the output control.
3. Capability to control the data "on" and "off" the system bus, independent of the device selecting function identified above.
4. Access time compatible with the high performance microprocessors that are currently available.

Now let's examine the system architecture that is required to implement the two line control and prevent bus contention. This is shown in the form of a timing diagram (Figure 5). As before, addresses are used to generate the unique device selecting function, but a separate and independent Output Enable (OE) control is now used to gate data "on" and "off" the system data bus. With this scheme, bus contention is completely eliminated as the processor determines the time during which data must be present on the bus and then releases the bus by way of the Output Enable line, thus freeing the bus for use by other devices, either memories or peripheral devices. This type of architecture can be easily accomplished if the memory devices have two control functions, and the system is implemented according to the block diagram shown in Figure 6. It differs from the previous block diagram (shown in Figure 2) in that the control bus, which is connected to all memory Output Enable pins, provides separate and independent control over the data bus. In this way, the microprocessor is always in control of the system; while in the previous system, the microprocessor passed control to the particular memory device and then waited for data to become available. Another way to look at it is, with a single control line the system is always asynchronous with respect to microprocessor/memory communications. By using two control lines, the memory is synchronized to the processor.

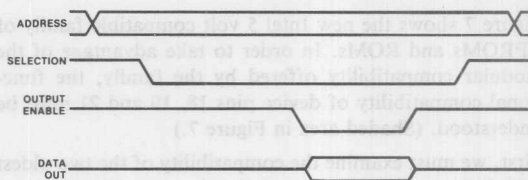


Figure 5. Two Control Line Architecture

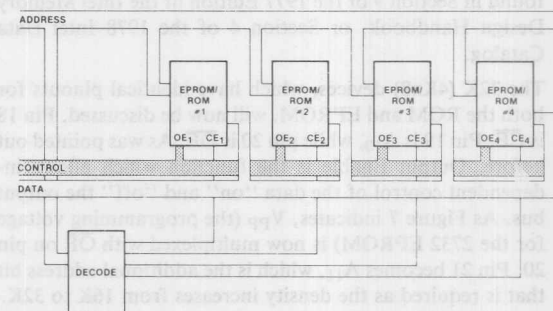


Figure 6. Two Control Line Architecture

TERMINOLOGY

Some of the terminology applied to the functions of the Intel 5 volt compatible family may be confusing or unfamiliar to many EPROM/ROM users, so the various terms are defined here. Actually, the nomenclature was developed by various standards groups and is reiterated here to avoid confusion as we begin a detailed discussion of the devices themselves.

First of all, Chip Enable (CE) must be defined, as it is the primary device selection pin. By agreed standards, that function which substantially affects power dissipation is called CE. Any memory device that has a CE function has both an active and standby power level associated with it.

Output Enable (OE) is the signal that controls the output. The fundamental purpose of OE is to provide a completely separate means of controlling the output buffer of the memory device, thereby eliminating bus contention.

Chip Select (CS) is a signal that gets logically ANDed with addresses. In a completely static device, CS must remain stable throughout the entire device cycle, and its function is equivalent to Output Enable (OE).

THE NEW INTEL FAMILY

Figure 7 shows the new Intel 5 volt compatible family of EPROMs and ROMs. In order to take advantage of the modular compatibility offered by the family, the functional compatibility of device pins 18, 19 and 21 must be understood. (Shaded area in Figure 7.)

First, we must examine the compatibility of the two oldest EPROM members of the 5 volt family — the 8K (2758) and the 16K (2716).

Pin 21 (V_{PP}) is normally connected to V_{CC} for read only applications of both devices, and pin 19 is either at GND (V_{IL}) for the 8K 2758 or connected to A_{10} for the 16K 2716. Further details on either of these devices can be found in Section 9 of the 1977 Edition of the Intel Memory Design Handbook, or Section 4 of the 1978 Intel Data Catalog.

The 32K (4Kx8) devices, which have identical pinouts for both the ROM and EPROM, will now be discussed. Pin 18 is \overline{CE} . Pin 19 is A_{10} , while pin 20 is \overline{OE} . As was pointed out before, Output Enable is the function which allows independent control of the data "on" and "off" the output bus. As Figure 7 indicates, V_{PP} (the programming voltage for the 2732 EPROM) is now multiplexed with \overline{OE} on pin 20. Pin 21 becomes A_{11} , which is the additional address bit that is required as the density increases from 16K to 32K.

Pin 21 is the only pin that requires any special consideration when designing a system to accept the 8K, the 16K, or the 32K device. With the 8K and the 16K devices, pin 21 must be connected to V_{CC} , while with the 32K and higher density devices, it must be connected to A_{11} . This is easily accomplished by making sure the printed circuit trace links all pin 21's together as though they were an address line and allowing for a jumper that will connect pin 21 to either V_{CC} or A_{11} at the edge of the array (this technique can be seen in the "Printed Circuit Board Design" section and in Figure 8). Connecting the pin 21's together in this manner is acceptable as the read current requirement for V_{PP} is 4mA maximum per device — low enough to be handled by a signal trace, but too high for an address driver to provide directly.

The highest density member of the family is a 64K ROM which is also shown in Figure 7. In order to maintain total compatibility it is packaged in a standard 28-pin package.

It may seem as though the 28 pin package is not compatible with the rest of the family, but referring again to Figure 7, note that the lower 24 pins are identical to the 24 pin 8K, 16K and 32K devices. To allow for total compatibility within the family: printed circuit boards must be laid out to accommodate 28 pin sites; a jumper must be included to accommodate pin 21 as shown in Figure 8, and when using 64K devices, CS_2 (Pin 26) must be mask coded active high. This compatibility can also be seen graphically in Figures 9 and 10. The upper portion of the figure shows how 24 pin devices are used in the 28 pin sites. The two control lines (\overline{CE} and \overline{OE}) remain unchanged as discussed earlier, and A_{12} , the next address bit required for a 64K bit device, is connected to pin 2 of the 28 pin site. The lower portion of the figure illustrates the use of 28 pin devices. Address bit A_{12} is already connected to the right pin, and the chip selects (CS_1 and CS_2) are connected to the V_{CC} power distribution grid. This configuration would require that both CS_1 and CS_2 be coded active high.

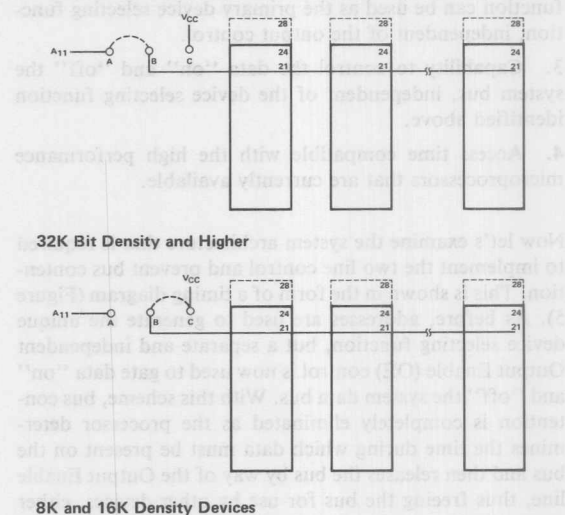


Figure 8. Pin 21 Connections for Various Density Devices

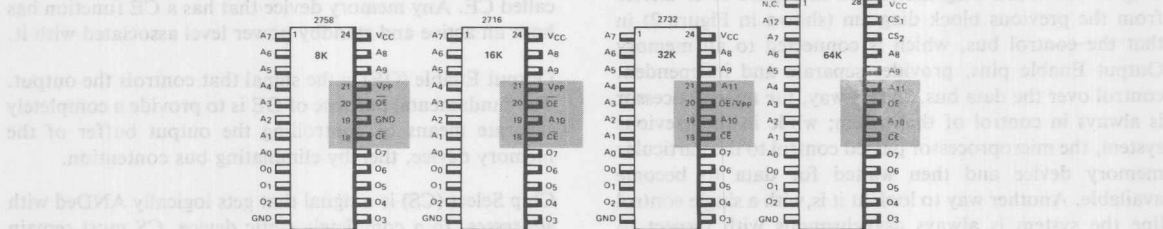


Figure 7. 5 Volt EPROM/ROM Compatible Family

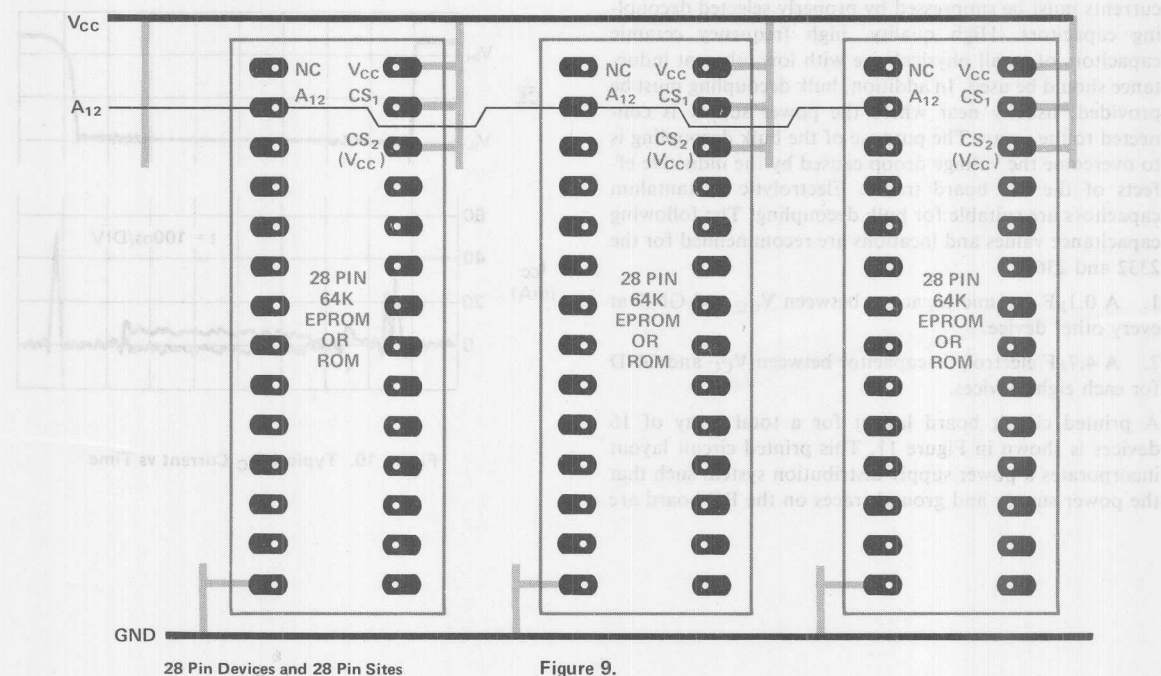
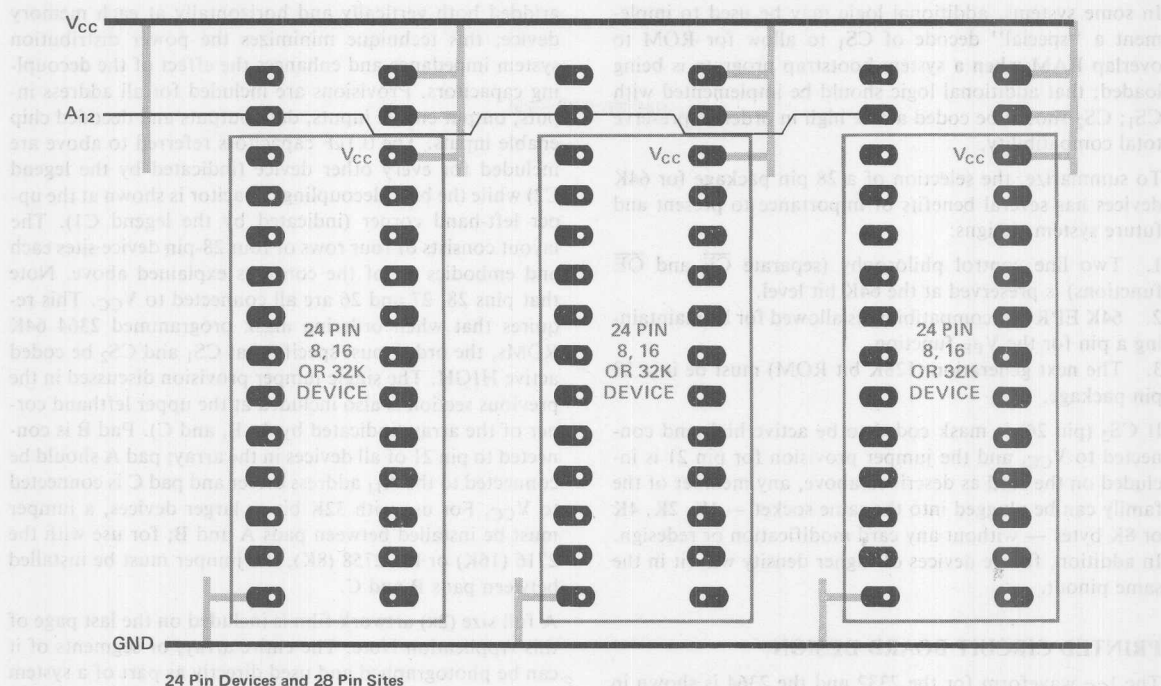


Figure 9.

Additional logic should be implemented with CS_1 ; CS_2 should be coded active high in order to preserve total compatibility.

To summarize, the selection of a 28 pin package for 64K devices has several benefits of importance to present and future system designs:

1. Two line control philosophy (separate \overline{CE} and \overline{OE} functions) is preserved at the 64K bit level.
2. 64K EPROM compatibility is allowed for by maintaining a pin for the V_{PP} function.
3. The next generation (128K bit ROM) must be in a 28 pin package.

If CS_2 (pin 26) is mask coded to be active high and connected to V_{CC} , and the jumper provision for pin 21 is included on the card as described above, any member of the family can be plugged into the same socket — 1K, 2K, 4K or 8K bytes — without any card modification or redesign. In addition, future devices of higher density will fit in the same pinout.

PRINTED CIRCUIT BOARD DESIGN

The I_{CC} waveform for the 2332 and the 2364 is shown in Figure 10. The supply current, I_{CC} , has three segments that are of concern to the system designer — the standby level, active level and the transient peaks that are produced on the rising and falling edges of Chip Enable. The transient currents must be suppressed by properly selected decoupling capacitors. High quality, high frequency ceramic capacitors of small physical size with low inherent inductance should be used. In addition, bulk decoupling must be provided, usually near where the power supply is connected to the array. The purpose of the bulk decoupling is to overcome the voltage droop caused by the inductive effects of the PC board traces. Electrolytic or tantalum capacitors are suitable for bulk decoupling. The following capacitance values and locations are recommended for the 2332 and 2364:

1. A $0.1\mu F$ ceramic capacitor between V_{CC} and GND at every other device.
2. A $4.7\mu F$ electrolytic capacitor between V_{CC} and GND for each eight devices.

A printed circuit board layout for a total array of 16 devices is shown in Figure 11. This printed circuit layout incorporates a power supply distribution system such that the power supply and ground traces on the PC board are

ing capacitors. Provisions are included for all address inputs, output enable inputs, data outputs and decoded chip enable inputs. The $0.1\mu F$ capacitors referred to above are included for every other device (indicated by the legend C2) while the bulk decoupling capacitor is shown at the upper left-hand corner (indicated by the legend C1). The layout consists of four rows of four 28-pin device sites each and embodies all of the concepts explained above. Note that pins 28, 27 and 26 are all connected to V_{CC} . This requires that when ordering mask programmed 2364 64K ROMs, the order must specify that CS_1 and CS_2 be coded active HIGH. The single jumper provision discussed in the previous section is also included at the upper lefthand corner of the array (indicated by A, B, and C). Pad B is connected to pin 21 of all devices in the array; pad A should be connected to the A_{11} address driver and pad C is connected to V_{CC} . For use with 32K bit or larger devices, a jumper must be installed between pads A and B; for use with the 2716 (16K) or the 2758 (8K), the jumper must be installed between pads B and C.

A full size (2x) artwork film is included on the last page of this Application Note. The entire array, or segments of it can be photographed and used directly as part of a system board.

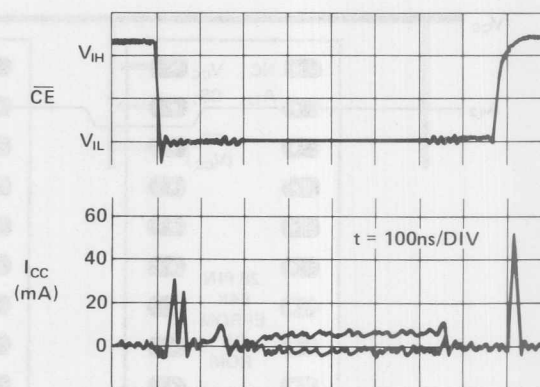
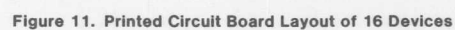


Figure 10. Typical I_{CC} Current vs Time



March 1978

CRYSTALS: Specifications for intel® Components

Bruce McCormick
Microcomputer Applications

Specifications for Intel Components

Introduction	13-3
Crystal Operation — Brief Theoretical Explanation	13-3
Circuit Configurations From Various Manuals/Explanation	13-4
Series 10 pF Capacitor	13-4
Parallel 20 pF Capacitor to Ground	13-4
8224 Overtone Application (Tank Circuit)	13-4
Precise Timing Applications	13-5
What If I Use a Crystal Other Than Specified	13-5
Specifications	13-5
Intel Component Crystal Requirements	13-5
Suggested Suppliers, Part Numbers	13-6

INTRODUCTION

The following brief note is intended to answer the simpler questions on crystal specifications and their operation with the various Intel components. First, a theoretical explanation of the crystal is given to aid the user in understanding crystal operation. This includes a discussion of the parameters necessary for proper specification to the vendor. Following this section are explanations of the various crystal-capacitor configurations seen in the Intel User's Manuals and data sheets; why they are suggested for proper crystal operation and what might happen if they weren't there.

The final section of this note provides a list of suggested crystal specifications, suppliers, and part numbers for the highest frequency crystals possible for the various Intel components that require them. In no way does this list represent the only crystals or suppliers available. This section is conveniently preceded by a discussion of problem areas that may result if a user is using the wrong crystal required for the component.

CRYSTAL OPERATION — BRIEF THEORETICAL EXPLANATION

Understanding Crystal Operation

Crystals are piezoelectric devices which transform voltage energy to mechanical vibrations and voltage oscillations. The frequency of the crystal is largely dependent on its thickness, with thinner crystals producing a higher frequency.

Crystals are generally specified as being series or parallel resonant, but all crystals are in actuality both. Vendors supply crystals as series or parallel resonant based on the desired frequency and the crystal's relative ability to generate the frequency in that mode. On a conceptual basis, when using a crystal as series resonant, its output is in phase with its input, whereas using the crystal as parallel resonant will result in a phase shift from its input to output.

Different LSI components prefer different crystals due to the nature of their internal oscillator design. In general, Intel bipolar components have a non-inverting, bidirectional drive oscillator, whereas NMOS components use an inverting oscillator. Non-inverting oscillators prefer series resonant crystals (as the series resonant crystal has 0 degree net phase shift), while inverting oscillators prefer crystals which are parallel resonant. Since a crystal has both a series and parallel operating frequency, many times any crystal will seem to work when connected to a component.

When giving the specifications to a crystal vendor for a crystal, it is helpful to understand its equivalent circuit as shown in Figure 1. The impedance of this circuit (neglecting R to simplify matters for conceptual purposes) can be calculated and plotted against frequency (Figure 2). This frequency-impedance plot illustrates the two different operating modes of crystal. ω_s (series resonance) occurs when the impedance (reactance) is zero and ω_p (parallel resonance) occurs when the impedance goes to infinity and appears inductive.

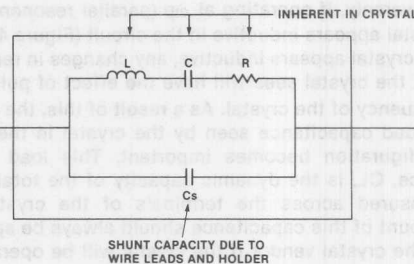


Figure 1

$$Z = \frac{(1/SC + SL) 1/SC_s}{1/SC + SL + 1/SC_s} = \frac{-j(\omega^2 - \omega_s^2)}{\omega C_s(\omega^2 - \omega_p^2)} \quad \text{WHERE } \omega_s = 1/\sqrt{LC} \quad \omega_p = 1/\sqrt{L(C_s(C + C_s))}$$

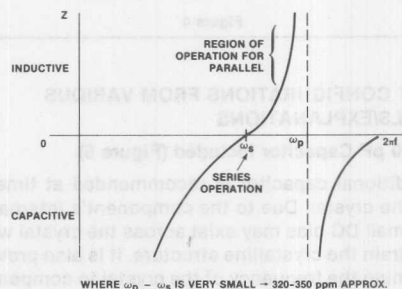


Figure 2

When operating at series resonance (ω_s) the equivalent circuit of the crystal becomes a simple resistor R_s (Figure 3; remember, R was neglected in the impedance calculation). This R_s value must be specified to the crystal vendor when buying a crystal.

This parameter becomes a problem with lower frequency or overtone crystals (thicker, more resistance) and a buffer that doesn't have sufficient gain to drive those crystals (i.e., loop gain becomes less than 1). Overtone crystals also have R_s problems as their R_s is associated with the fundamental frequency of the crystal, not the 3rd harmonic or overtone. The 8224 is particularly sensitive to R_s with 27 MHz overtone applications.



Figure 3

Conversely, if operating at ω_p (parallel resonance), the crystal appears inductive in the circuit (Figure 4). Since the crystal appears inductive, any changes in reactance that the crystal sees will have the effect of pulling the frequency of the crystal. As a result of this, the amount of load capacitance seen by the crystal in the circuit configuration becomes important. This load capacitance, CL , is the dynamic capacity of the total circuit measured across the terminals of the crystal. The amount of this capacitance should always be specified to the crystal vendor if the crystal will be operating at parallel resonance.

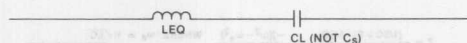


Figure 4

CIRCUIT CONFIGURATIONS FROM VARIOUS MANUALS/EXPLANATIONS

Series 10 pF Capacitor Included (Figure 5)

This additional capacitor is recommended at times to debias the crystal. Due to the component's internal circuit, a small DC bias may exist across the crystal which would strain the crystalline structure. It is also provided for trimming the frequency of the crystal to compensate for the loading effects of the component.

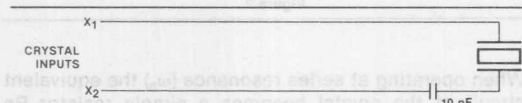


Figure 5

Parallel 20 pF Capacitors to Ground (Figure 6)

Crystals can oscillate at several different frequencies, each emanating from a different direction of vibration in the crystal. For a crystal to oscillate during startup in its fundamental frequency, it is best for the crystal to see the slew rate (Figure 7) of the pulse provided from the oscillator to be as close to the operating frequency as possible.

These 20 pF capacitors act as a high frequency filter to create a slew rate closer to the fundamental frequency of the crystal. As can be guessed, lower frequency crystals are more susceptible to the problem of not starting up in the fundamental frequency.

Capacitors are placed on both sides of the crystal as some components have bidirectional drive buffers (i.e., 1/2 of cycle drive from one side, other half from opposite side). A crystal that needs these extra 20 pFs to ground will be characterized by starting up at a 3rd or 5th harmonic instead of the fundamental frequency. The CL specifications in the specification section takes into consideration these extra 20 pF capacitors required for some Intel components for proper operation.

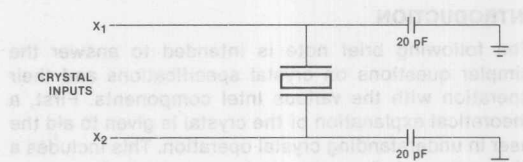


Figure 6

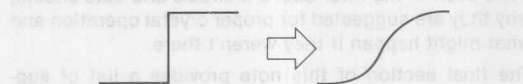


Figure 7

Tank Circuit

On some Intel components, provision is made for a tank circuit. This is for the use of an overtone crystal; i.e., one that is working at a harmonic (generally its 3rd). The tank circuitry is a filter to bypass the lower and higher, unwanted frequencies to ground while appearing "open" to the desired frequency. It is necessary to use tank circuits and overtone crystals when in the 25+ MHz range and above. Fundamental crystals are difficult to make in this frequency range as the crystal must be thinner for higher frequencies.

A circuit that has been used for the 8224 in 27 MHz overtone crystal applications is shown in Figure 8.

This filter can be approximated through formulas where afterwards it will be necessary to tweak the component values for optimization. The formula used to get the original component values is:

$$f = \frac{1}{2\pi\sqrt{L_1 C_1}} \quad \text{where } f = \text{overtone frequency}$$

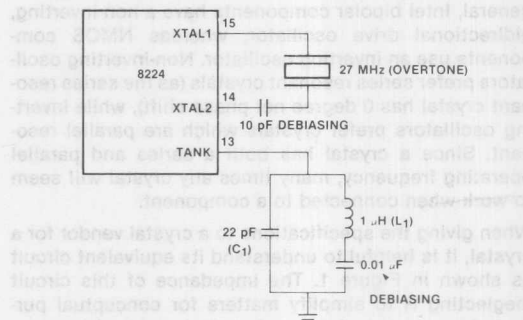


Figure 8

Precise Timing Applications

For applications where precise timing is required, using an external drive could produce better results. The accuracy of the component clock over temperature will be as accurate as the external drive. It is difficult to guarantee the temperature stability of the output frequency of the Intel component as fabrication process parameters vary, causing large ranges of input impedance and hence a large range of loading for the crystal.

WHAT IF I USE A CRYSTAL OTHER THAN SPECIFIED?

Series vs. Parallel

As discussed in the theoretical section, all crystals have a series and parallel operating mode. Placing a series crystal on a device requiring a parallel (i.e., there is an inverting oscillator between the two inputs) will force the crystal to oscillate in its parallel mode (and vice versa). A system with the wrong crystal will exhibit its clock frequency shifted a small percentage (about 320 to 350 parts per million) from the specified crystal frequency. When using the wrong crystal, any attempts to trim the frequency to its specified value by using small parallel (series if series crystal) variable capacitors will cause the crystal to stop oscillating, as predicted by theory. If the correct crystal is being used, trimming can be done.

In applications where accuracy is not important, series crystals are sometimes substituted for parallel in the circuit. For instance, the 8048 has been characterized to be compatible with the series color burst TV crystal

(3.579545 MHz). If this crystal is used, a small frequency shift will occur, as noted above.

Insufficient Drive Level

The drive level specified is the maximum amount of power that is expected for the crystal to dissipate. If the crystal can't handle this level, frequency drift may occur or possible fracture of the crystal. In other words, if the crystal used cannot handle the oscillator drive level, long term reliability problems may occur.

Rs Too High

The higher Rs is, the higher the drive capability of the oscillator has to be to get the crystal to oscillate. Too much Rs may result in the oscillator not being able to drive the crystal; i.e., the loop gain is less than one. Overtone applications are particularly sensitive to this as thicker crystals are used (lower fundamental frequency, more resistance).

SPECIFICATIONS

Intel Component Crystal Requirements

The following is a list of suggested specifications for crystals to be used with Intel components. In most instances the upper frequency limit is given, with exceptions being footnoted.

Component (Function)	Process	Component Divide By	Crystal Type	Fundamental Overtone	Upper Limit Frequency
1. 4201A (Clock Generator)	CMOS	—	Series	f	5.185 MHz
2. 8035/48/49, 8748 (8-Bit CPU)	NMOS	15	Parallel	f	6.0 MHz
3. 8748/8035-8 (8-Bit CPU)	NMOS	15	Parallel	f	3.6 MHz
4. 8041/8741 (Universal Peripheral Interface)	NMOS	15	Parallel	f	6.0 MHz
5. 8085A (8-Bit CPU)	NMOS	2	Parallel	f	6.25 MHz/6.144 MHz ⁽¹⁾
6. 8085A-2 (8-Bit CPU)	NMOS	2	Parallel	f	10.0 MHz
7. 8202 (Dynamic RAM Controller)	Bipolar	—	Series	f	25 MHz
8. 8224 (8080A Clock Generator)	Bipolar	—	Series	f/o	27 MHz/18.432 MHz ⁽²⁾
9. 8284 (8086 Clock Generator)	Bipolar	3	Series	f	24 MHz/15 MHz ⁽³⁾

Additional suggested specifications:

Frequency Tolerance:	± 0.005% (up to the user)
CL (Load Capacitance):	= 20-35 pF (not necessary when specifying series)
Rs (Equivalent Series Resistance):	<75 ohms
Cs (Shunt Capacitance):	<7 pF
Drive Level:	<10 MHz crystal 10 milliwatts
	>10 MHz crystal 5 milliwatts

- Notes:**
- 6.144 MHz is commonly used as convenient baud rates can be generated from this frequency.
 - 27 MHz is max. 18.432 is common crystal used which gives maximum clock rate for 8080A. Fundamental crystal should be used for the 18.432 MHz application.
 - Used for either a 8 or 5 MHz output clock, respectively.

Holder specifications are up to the user. A standard popular one that provides ample lead length is HC-33/U (0.750"W x 0.765"H, 1.5" lead length with spacing of 0.486") and can be used for frequencies up to 4 MHz. After 4 MHz a smaller holder can be used such as HC-18/U (0.435"W x 0.530"H, 1.5" lead length with spacing of 0.192"). All crystals listed in the following table will fit in the HC-33/U holder. Other standard holders are available.

Suggested Suppliers, Part Numbers

The following are two vendors (which are among many) that supply crystals to the specifications given earlier and their part numbers (given in order of frequency). The user should make sure that the holder type associated with these part numbers is acceptable in their application.

f	Parallel/ Series	Crystek ⁽¹⁾ Corp.	CTS Knight, ⁽²⁾ Inc.
3.6 MHz	P	**	**
5.185 MHz	S	CY8A	**
6.0 MHz	P	**	MP060
6.144 MHz	P	**	MP061
6.25 MHz	P	**	MP062
10.0 MHz	P	**	MP10A
15.0 MHz	S	CY15A	MP150
18.432	S	CY19B*	MP184*
24.0 MHz	S	**	MP240
25.0 MHz	S	**	MP250
27.0 MHz	S (overtone)	CY27A	MP270

*Intel also supplies a crystal numbered 8801 for this application.

**Contact vendor with the appropriate specifications.

Notes: 1. Address: 1000 Crystal Drive, Fort Meyers, Florida 33901
2. Address: 400 Reimann Ave., Sandwich, Illinois

The user is not limited to these vendors or frequencies. The frequency chosen by the user should take into consideration convertibility to desired baud rates and the system timings that must be met.

In summary, to obtain a crystal for the user's application, it is necessary to give the crystal vendor the following information:

Series or parallel
Fundamental or overtone
Rs (series), Cs (shunt)
CL if parallel
Drive Level
Frequency tolerance
Holder type

For a select few crystals, vendor numbers were given for two different vendors. With the above information, most vendors can make the desired crystal whether or not they have it as a standard part.

Component (Function)	Process	Drives By	Component Type
1. 8201A (Clock Generator)	CMOS	—	—
2. 8201B (8-Bit CPU)	HMOS	15	Parallel
3. 8201C (8-Bit CPU)	HMOS	15	Parallel
4. 8201D (Universal Peripheral Interface)	HMOS	15	Parallel
5. 8201E (8-Bit CPU)	HMOS	15	Parallel
6. 8201F (8-Bit CPU)	HMOS	15	Parallel
7. 8201G (Dynamic RAM Controller)	Bipolar	—	Series
8. 8201H (8080A Clock Generator)	Bipolar	—	Series
9. 8201I (8080 Clock Generator)	Bipolar	15	Series

Additional suggested specifications:

Frequency Tolerance: ±0.005% (up to the user)
CL (Load Capacitance): = 20-30 pf (not necessary when specifying series)
Rs (Equivalent Series Resistance): <75 ohms
Cs (Shunt Capacitance): <7 pf
Drive Level: <10 mW crystal, 10 milliwatts
>10 mW crystal, 5 milliwatts

Notes: 1. 8201A-M is not a standard part. It is a custom part. It is not recommended for use in a standard system. 2. 8201H is a custom part. It is not recommended for use in a standard system. 3. 8201I is a custom part. It is not recommended for use in a standard system.